Sketching Slides

Interactive Creation and Automatic Solution of Constrained Document Layout Problems

DISSERTATION

zur Erlangung des akademischen Grades
doctor rerum naturalium
(Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
Herrn Dipl.-Inf. Sebastian Christoph Theophil
geboren am 28.11.1979 in Berlin

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Dr. h.c. Christoph Markschies

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Peter Frensch

Gutachter:

1. Prof. Dr. Hans-Dieter Burkhard
2. Dr. Markus Hannebauer
3. Prof. Irfan Essa, PhD

eingereicht am:	6.10.2010
Tag der mündlichen Prüfung:	18.2.2011

Abstract

The efficiency of desktop publishing is severely limited by the lack of sophisticated automatic document layout systems. State-of-the-art algorithms either require the input to be written in a description language such as HTML and LaTeX, or to be a manually designed layout template. However, description languages are ill-suited to express layout problems with weak semantics and layout templates shift the burden from the end user to the template designer.

This thesis defines a general layout problem with linear constraints in simple geometric terms. This problem definition encompasses many well-researched layout problems, including table layout problems, yellow page layout problems, and many user interface layout problems.

The first contribution of this thesis is an algorithm that solves this general class of layout problems by treating them as equitable resource allocation problems. The available document area is a resource that is distributed among inter-element gaps. The layout problem is transformed into a lexicographic min-ordering optimization problem that is solved using linear programming techniques in real-time. User-generated input problems are frequently over- or under-constrained. If the layout problem is over-constrained, the quality of the solution layout degrades gracefully. The layout algorithm finds the solution layout with the most equitable distribution of constraint errors among the soft layout constraints, i.e., the solution closest to the user's original intent. Conversely, the layout algorithm detects the under-constrained subproblems that adversely affect the solution layout. It adds the minimal number of constraints required to achieve the fully specified layout problem that is closest to the user's input.

The second contribution of this thesis is the creation of an intuitive direct manipulation user interface that lets users create the aforementioned class of general constrained layout problems. It hides the complexity of the constraint system and avoids the usability problems that have plagued constraint drawing applications. It eliminates the need of document description languages and manually-created layout templates.

The layout algorithm and the user interface have been implemented in our ICBM system. In the evaluation, we show that the best state-of-the-art specialized table layout algorithms do not outperform the general ICBM layout algorithm by any significant margin.

Keywords:
Automatic Layout, Constraint Drawing, Interaction, Optimization

Zusammenfassung

Die Entwicklung effizienter Desktop Publishing Systeme wird behindert durch den Mangel an leistungsfähigen, automatischen Layoutalgorithmen. Aktuelle Algorithmen zum Layout ganzer Dokumente oder einzelner Seiten erfordern entweder die Formulierung des Layoutproblems in einer formalen Beschreibungssprache, oder sie benötigen fertige, detaillierte Layouttemplates. Layoutprobleme mit schwacher Semantik lassen sich schlecht in formale Sprachen umsetzen, Layout Templates verschieben den manuellen Aufwand nur vom Endnutzer zum Template Designer.

Diese Dissertation definiert ein allgemeines, geometrisches Layoutproblem mit linearen Constraints. Dieses Problem umfasst verschiedene, gut untersuchte Layoutprobleme, z.B. Tabellenlayout, Yellow Page Layout, und User Interface Layout Probleme.

Das erste Ergebnis dieser Dissertation ist ein Layoutalgorithmus, der das beschriebene Layoutproblem löst, in dem er es als Ressourcenallokationsproblem interpretiert. Die Fläche einer einzelnen Seite ist eine Ressource, die zwischen den visuellen Elementen einer Seite verteilt wird. Das Layoutproblem wird in ein lexikographisches min-ordering Optimierungsproblem übersetzt, das durch lineare Optimierung in Echtzeit gelöst wird. Die Lösungen manuell erzeugter Layoutprobleme sind häufig über- oder unterbestimmt. Wenn das Problem überbestimmt ist, also keine gültige Lösung besitzt, muss der Algorithmus die Lösung finden, die am nächsten an der intendierten Lösung ist. Der Algorithmus erkennt nicht eindeutig definierte Probleme mit unbefriedigenden Lösungen und fügt die minimal notwendige Anzahl von Constraints hinzu um das vom Nutzer beabsichtigte Layout zu erzeugen.

Das zweite Ergebnis dieser Dissertation ist die Entwicklung einer intuitiven Benutzerschnittstelle, die es erlaubt, die vorhergehend beschriebenen Layoutprobleme zu erzeugen. Sie verbirgt die Komplexität des Constraintsystems und vermeidet die Komplexität constraint-basierter Grafikanwendungen der Vergangenheit. Diese Benutzerschnittstelle macht formale Beschreibungssprachen und manuell erzeugte Layouttemplates überflüssig.

Der Layoutalgorithmus und die Benutzerschnittstelle wurden als Teil des ICBM Systems implementiert. Die Evaluation zeigt, dass die besten Tabellenlayoutalgorithmen keine signifikant besseren Ergebnisse produzieren als der allgemeinere ICBM Layout Algorithmus.

Schlagwörter:
Automatisches Layout, Constraints, Interaktion, Optimierung

To my family, Ines and Sophia.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When the creative part of writing a new and insightful text has been finished, the chore of adapting the document to its desired output medium begins. Today, a single illustrated text may be printed as a book, published on a website or transformed into a screen presentation in front of a large audience. Each output medium has different requirements on the size of the font and the images as well as on the content's arrangement. Put differently, each medium imposes different constraints how the document must be presented that are completely independent of the document content.

Many researchers have developed applications that allowed the user to separate the document content from its graphical presentation. These applications derive constraints from the chosen output format and calculate the optimal document layout given the user-defined content and the format-specific constraints. Other applications derive constraints from the semantics of the document content itself.

Generating document layouts without any user intervention can be very convenient but the user loses all control over the result. Few people would notice if the latest quarterly SEC filing had been laid out automatically. A print magazine editor on the other hand would never accept anything less than total control to achieve a pixel-perfect layout.

In general, high-quality document layouts can be generated automatically if the document structure and content strongly suggest a certain graphical arrangement. The less the content is structured, the more user input is necessary to find the visual arrangement that corresponds to the content's semantics. As an example, consider the layout of presentation slides. Often, they contain disparate pieces of information that are held together by what the presenter is saying while the slide is shown. Such a slide cannot be laid out automatically without knowledge of the oral presentation. Even if there were a simple way for the user to express her thoughts in a form that a

document layout system could interpret, the user would find this hard to do without actually seeing the layout. An interactive application can help the user discover how her argument can be brought across visually but also what her argument actually is. Being able to try out and discard alternative layouts can guide the user towards a more precise understanding of what exactly the slide should express.

The user's primary task is to develop the global layout structure: Which element is drawn on top, or on the left, where it is first read? Which elements are presented adjacent to that, below or next to it? Charts, tables, images, text boxes are arranged by the user in a way that clearly expresses how the presented pieces of information complement each other. A layout system can aid this process by taking care of the details. If the user squeezes another chart between two explanatory boxes of text, both must be moved apart to accommodate the added content. If the user types into a text box, its size must increase, and neighboring elements have to move.

So far, little work has been undertaken to make the creation of adaptable constraint-based document layouts an interactive process. There is an abundance of special layout tools for state diagrams, flow charts, and organizational charts that exploit application specific constraints. But by their very nature, special purpose layout tools are not flexible and they limit the user to one specific presentation format. On the other hand, a user can quickly sketch diagrams with a standard drawing application, but once he or she tries to make further changes to the document it becomes apparent where these programs fall short: General purpose drawing programs are unaware of diagram specific layout constraints, e.g., that shapes in the same hierarchy level should be aligned and have the same size. A small change to one part of the drawing can lead to a cascade of necessary adaptations until the desired layout is reestablished.

The thesis of my dissertation is that with the computer performance available today, user interactions can be analyzed, visualized, and transformed into a system of layout constraints that can be solved in real-time using a generic layout algorithm. The resulting system unites the adaptability of constrained layouts with the familiarity and flexibility of direct manipulation interfaces, and it can be used to create an unprecedented variety of document layouts.

1.1 Contributions

The contributions of this dissertation are two-fold:

1. This thesis presents a layout algorithm that calculates solutions for a very general class of layout problems.

   - A layout problem is defined over so-called *gridlines*. A gridline represents, e.g., the left coordinate of a text box.
   - Gridline relations may be further specified using linear constraint equations. Linear constraints allow the specification of absolute gridline positions, but the layout algorithm does not require any positional information.
   - The presented layout algorithm solves this general class of layout problems without further information using an equitable resource allocation approach.
   - It is capable of handling under-constrained and over-constrained input problems gracefully. Its output is deterministic and aesthetically pleasing in case of under-constrained inputs and it minimizes the extent of infeasibilities if the input has been over-constrained.

   The layout algorithm solves an unprecedentedly general class of layout problems. The class of permitted problems encompasses the problems of table layout, flow chart layout, and general page layout. The presented algorithm can be used to relayout pages when the page size changes. The layout algorithm is compared to several special-purpose table layout algorithms and the evaluation shows that it calculates results of equal quality on a majority of problem instances, despite its superior generality.

2. The second contribution of this thesis is the direct manipulation user interface designed to let users specify the aforementioned class of layout problems.

   - The user interface reuses familiar interaction mechanisms of well-known applications like Microsoft PowerPoint. It lets users interact directly with graphical objects like text boxes, arrows, and charts to build page layouts. The user interface never exposes the underlying constraint system to the user.
   - All user actions are continuously visualized and can be be easily reversed in case of an error, which lets the user feel in control of the system.

- By default, all user actions are implicitly translated into document layout constraints that express the user's expectations. Slide elements are implicitly constrained to fit their text content. The discrete set of narrowest width-height-configurations of each text-containing shape is calculated using a Dynamic Programming algorithm. The step-function formed by these width-height-configurations is approximated using linear equations.
- If desired, the user has total control over the resulting layout because she can specify object positions and sizes explicitly. These interactions are familiar to users of interactive drawing programs and do not require the user to understand the concept of constraints. The user interface clearly divides the responsibilities of the user and the layout system. It is always either the layout algorithm or the user who specifies object positions and sizes. There can never be a conflict between both.
- The presented interface includes most familiar interactions like cut, copy, paste, undo and redo which have been extended to manipulate both shapes and the constraints between them. All user interactions are built from a set of elementary operations to manipulate instances of the layout problem.

1.2 Outline

The following chapter starts with an introduction to constraint satisfaction and constraint optimization problems and how these problems have been applied to the domains of constrained drawing and automatic document layout. The chapter gives an extensive overview of the work that has been done in these two problem domains. From this body of research I derive a list of requirements that an interactive constraint applications must meet.

Chapter 3 illustrates with the help of a simple example the shortcomings of the current state-of-the-art in user interaction and document layout applications. It proceeds with a high-level overview of this thesis' proposed answer to these shortcomings, the so-called ICBM system. The chapter closes with a formal definition of the layout problem.

Chapter 4 describes a set of algorithms that can be used to create and manipulate instances of the layout problem defined in chapter 3. The algorithms describe how to insert, delete, move, and rotate shapes, their attached constraints, and the gridlines that the shapes are bound to. These algorithms can be used in the implementation of a user interface that transforms familiar interaction patters into instances of the layout problem.

Chapter 5 is divided in two parts. In the first part, I describe how to derive layout constraints from the user input, i.e., how to interpret the user's drawing. The layout algorithm derives constraints on the desirable text box sizes and constraints that govern the distribution of space among inter-element gaps. In the second part, I describe the layout problem as a lexicographic min-ordering optimization problem and I present an algorithm that solves this kind of linear optimization problem. The presented algorithm handles over-constrained problems by maximizing the degree of constraint satisfaction in soft constraints. A separate algorithm detects the undesired effects of an under-constrained problem a posteriori and adds additional constraints until the layout solution is uniquely defined.

Chapter 6 demonstrates the capabilities of the ICBM system. Carefully chosen example layouts show the generality and the capabilities of the presented layout algorithm. In a quantitative study, I compare the ICBM layout algorithm to the best state-of-the art table layout algorithms and I show that despite its superior capabilities, the ICBM layout algorithm performs as well as specially optimized table layout algorithms on the same input problems.

Chapter 7 presents a summary of the lessons learned from this work and an outlook on future improvements to the ICBM system.

Chapter 2

Background

The first application that featured a graphical user interface was Sketchpad, developed in 1964 by Ivan E. Sutherland [113]. Sketchpad was an interactive application to create geometric drawings with the help of constraints. The user could draw on the screen with a light pen. A row of switches next to the screen controlled the constraints. The user could, e.g., constrain two lines to be perpendicular by pressing the constraint switch and touching both lines with the pen. Sketchpad supported grouping primitive elements into object instances. Such compound objects could be duplicated, moved, and manipulated. Exploring the capabilities of his new tool, Sutherland noted two useful applications: Kinematics and Computer-Aided Design. Sutherland described a simple constrained animation shown in Fig. 2.1: Three lines, each of fixed length, are connected to form a chain. The first line is fixated at its beginning and the last line at its center. If the user moves the first line up and down on the screen, the third line starts to rotate like a steam engine drives a camshaft.

The idea to use constraints in graphical applications is so natural, that it exists for as long as graphical user interfaces do. Over the last forty years constraints have been applied to many different application scenarios, including user interface design, geometrical drawing, graph layout, table layout, and document layout. By defining a constraint, a designer can express the relation he wants two graphical objects to satisfy without having to define how this relation is to be maintained. Specialized solution algorithms exist for many classes of constraint problems.

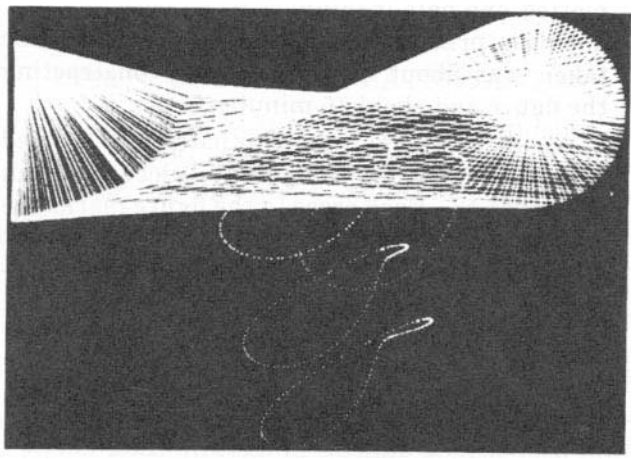


Figure 2.1: Three linked lines animated interactively in SketchPad. Image originally appeared in [113].

2.1 Constraint Satisfaction

A constraint satisfaction problem consists of a set of variables, each of which represents a decision that can be made. Associated with every variable is a domain of values that this variable can take. Deciding on a value for one variable can have implications for the remaining decision variables. These consequences are modeled by constraints. Each constraint restricts the values that can be assigned to a set of variables.

Definition 1. A **Constraint Satisfaction Problem (CSP)** is defined by a pair $\Pi_{\text{sat}} = (X, C)$ where

- $X = \{(v_1, D_1), \ldots, (v_n, D_n)\}$ is a set of domain variables $x_i = (v_i, D_i)$ each of which comprises a variable v_i and a domain D_i. $\Delta(\Pi_{\text{sat}}) = D_1 \times \cdots \times D_n$ is called the search space of Π_{sat}.
- $C = \{c_1, \ldots, c_m\}$ is a set of constraints $c_j \subseteq \Delta(\Pi_{\text{sat}})$.

The set

$$S(X, C) = \{(d_1, \ldots, d_n) \in \Delta(\Pi_{\text{sat}}) \,|\, \forall c_i \in C : (d_1, \ldots, d_n) \in c_i\}$$

is called solution space of Π_{sat}. Every $(d_1, \ldots, d_n) \in S$ is called a consistent solution of Π_{sat}, i.e., every variable v_i is assigned a value $d_i \in D_i$ that satisfies every constraint in C.

A variable's domain can be a set of discrete values such as a set of colors $\{black, white\}$ or a continuous set of values such as the set of all real values $\mathbb{R}$. Constraint satisfaction problems only answer the question if a solution satisfying all the given constraints exists. Often, many such solutions exist and the real problem is deciding which one of those is best.

Definition 2. A **Constraint Optimization Problem (COP)** is defined by a triple $\Pi_{\text{opt}} = (X, C, o)$ where X and C are defined as in Def. 1 and $o : \Delta(\Pi_{\text{opt}}) \to \mathbb{R}$ is a function that assigns each variable assignment from the underlying CSP a real value corresponding to its desirability. o is called *optimization criterion* or *objective function.*

Usually, constraints are defined implicitly, i.e., in terms of equations, and not as an explicit enumeration of permitted values. Many constraint satisfaction and constraint optimization algorithms exist that find a solution to such an implicitly defined CSP or COP. Several papers survey the constraint formulations typically used in constraint-graphics applications and their associated solution algorithms [6, 63, 64, 120]. For completeness, these kinds of constraints and their solution algorithms are presented on the following pages.

2.1.1 One-way Constraint Propagation

One-way or data flow constraints are a simple and common class of constraints [119]. A one-way constraint is an equation of the form

$$y = F(p_1, \ldots, p_n) \qquad y \in D_0, p_1 \in D_1, \ldots, p_n \in D_n$$

where the p_i are called parameters and $F : D_1 \times \cdots \times D_n \to D_0$ is a user-supplied function. Every one-way constraint only has a single left-hand variable y.

Definition 3. A pair $\Pi = (X, C)$ with

$$X = \{(x_0, D_0), \ldots, (x_n, D_n)\}$$

and

$$C = \{c_i : x_{i_0} = F_i(x_{i_1}, \ldots, x_{i_m}) \,|\, F_i : D_{i_1} \times \cdots \times D_{1_m} \to D_{i_0}\}.$$

is called a **One-Way CSP** if and only if every connected component of the *one-way constraint graph* $G(\Pi) = (V, E)$ with the set of nodes $V = \{x_0, \ldots, x_n\}$ and the set of directed edges

$$E = \bigcup_{c_i \in C} \{(x_{i_1}, x_{i_0}), \ldots, (x_{i_m}, x_{i_0}) \,|\, c_i : x_{i_0} = F_i(x_{i_1}, \ldots, x_{i_m})\}$$

is a tree where each node has at most a single incoming edge.

The term *one-way* now describes a property of the propagation algorithm used to solve problem Π. If Π is a One-Way CSP, a one-way constraint satisfaction algorithm only has to propagate the variable values from each node along the directed edges of graph $G(\Pi)$ [31, 105, 58, 1, 5, 4]. The propagation is conflict-free, because every vertex is either a node with only outgoing edges or it only has a single incoming edge defining its value. Since $G(\Pi)$ is a tree, it has no circles and thus no path following the directed edges can visit a vertex twice.

One-way constraints and their solution algorithms have been developed to solve very simple problems arising in interactive applications in which the user can only manipulate a single variable at a time whose changes have to be propagated through the above graph.

2.1.2 Multi-Way Constraint Propagation

Many problems arising in interactive constraint applications cannot be expressed as One-Way CSPs. Multi-way constraint propagation algorithms solve a variety of constraint optimization problems that arise frequently in constraint drawing applications [107, 39, 118, 62]. If a user-defined constraint satisfaction problem is under-specified, i.e., its solution set is very large, it is necessary to decide upon the best solution according to some objective criterion. Conversely, if the solution set is empty, i.e., if no assignment of values to variables satisfies all constraints, it is often desirable to find the assignment that is closest to a consistent solution.

Definition 4. A **Hierarchical Constraint Optimization Problem** is defined as a quadruple $\Pi_{hcop} = (X, C, o, h)$ where the triple (X, C, o) is a constraint optimization problem according to Def. 2 and $h : C \to \{0, \ldots, m\}$ is a function that assigns each constraint in C a hierarchy level. The optimization criterion o is defined as a function $o : \Delta(\Pi_{hcop}) \to \mathbb{R}^m$ where for every assignment of values to variables $d \in \Delta(\Pi_{hcop})$:

$$o(d) = (\text{card}(\{c \,|\, h(c) = 0 \wedge d \in c\}), \ldots, \text{card}(\{c \,|\, h(c) = m \wedge d \in c\})).$$

Thus, the i-th component of vector $o(d)$ defines how many constraints in hierarchy level i d satisfies.

Definition 5. Given a hierarchical constraint optimization problem $\Pi = (X, C, o, h)$ according to Def. 4, a tuple $d \in \Delta(\Pi_{hcop})$ is called a *solution of* Π if

$$\neg\exists d' \in \Delta(\Pi_{hcop}) : o(d') >_{\text{lex}} o(d)$$

where $>_{\text{lex}}$ is the lexicographic $>$ operator.

According to this definition, a constraint in level i dominates the constraints of subsequent less important levels $j > i$. If a solution d satisfies the same number of constraints in levels 0 to i as solution d', but one more constraint in level $i+1$, the number of constraints satisfied by solution d' in levels $j > i+1$ is not considered anymore [17, 19, 15].

Instead of maximizing the number of satisfied constraints in each hierarchy level, a user might be interested in satisfying some constraints as much as possible.

Definition 6. A **Constraint Error Optimization Problem** is defined as a quintuple $\Pi_{\text{ecop}} = (X, C, o, h, e)$ where X, C, and h are defined as in Def. 4 and where $e : C \times \Delta(\Pi_{ecop}) \to \mathbb{R}$ is an error function that for each constraint $c \in C$ and solution $d \in \Delta(\Pi_{ecop})$ computes a measure of how far away d is from satisfying c. The optimization criterion o is defined as a function $o : \Delta(\Pi_{ecop}) \to \mathbb{R}^m$ where for every assignment of values to variables $d \in \Delta(\Pi_{ecop})$:

$$o(d) = (\sum_{c \in C, h(c)=0} e(c,d), \ldots, \sum_{c \in C, h(c)=m} e(c,d)).$$

As in Def. 5, the lexicographic maximum of $o(d)$ over all $d \in \Delta(\Pi_{ecop})$ is the solution to problem Π_{ecop}.

2.1.3 Linear Optimization Methods

In recent years, linear constraints have become the standard in constraint-based graphical applications. The following example shows a simple linear programming problem:

$$\begin{aligned} \max\ & x_2 \\ & x_1 + x_2 \leq 7 \\ & -2x_1 + x_2 \leq 4 \\ & -0.5x_1 + x_2 \leq 3 \\ & x_1, x_2 \in \mathbb{R} \end{aligned}$$

The problem is illustrated in Fig. 2.2. The three solid lines indicate the boundaries of the half planes defined by the inequalities in the above example. Their intersection, i.e., the problem's feasible region, is drawn in grey. The dashed line is the objective function that takes its maximum at the tip of the feasible region. The problem has a unique solution in point $(2\frac{2}{3}, 4\frac{1}{3})$.

More formally, a linear optimization problem is the problem of maximizing or minimizing a linear objective function, subject to linear equality or inequality constraints.

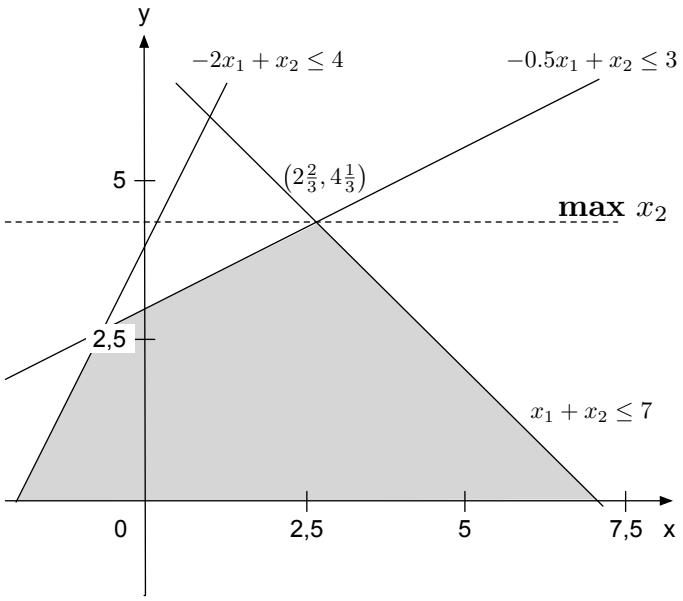


Figure 2.2: A simple linear programming problem with a unique solution.

Definition 7. A **Linear Optimization Problem** in canonical form is defined as

$$\begin{aligned} &\max c^T x \quad \text{subject to} \\ &\quad Ax = b \\ &\quad c \in \mathbb{R}^n, x \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n} \end{aligned}$$

where c is the vector of objective function coefficients and the equations $Ax = b$ are constraints.

The feasible region defined by the constraint equalities is a convex polyhedron. Consequently, any locally optimal solution is globally optimal. In the example of Fig. 2.2, the feasible region is a two-dimensional convex polygon.

A linear optimization problem does not have a solution if its feasible region is empty, i.e., if several constraints contradict each other. The feasible solution can be unbounded in the direction of the objective function and in this case there is no optimal solution since solutions can be constructed with infinitely large objective function values. If the problem is neither infeasible nor unbounded, the objective function always takes its maximum value in one of the polyhedron's vertices. An algorithm iterating over every vertex of the polyhedron will find the optimal solution. This solution is not necessarily unique. In fact, the solution is either unique or there are infinitely many

solutions that cover an edge or facet of the polyhedron [28]. If two points are optimal solutions, all their convex combinations must be optimal solutions as well and thus there can be zero, one, or an infinite number of optimal solutions to a convex optimization problem.

The simplex algorithm, developed by G. B. Dantzig in 1947, solves linear optimization problems by moving along a path of neighboring vertices such that with each step, the objective function improves or remains unchanged [29]. In practice, the simplex algorithm is very efficient, typically with time complexity $\mathcal{O}(m)$ where m is the number of constraints [22]. The algorithm's worst-case complexity, however, is exponential, as has been demonstrated by Klee and Minty using a constructed optimization problem [80].

Interior-point algorithms [78] solve linear programming problems in polynomial time by calculating a sequence of solutions on the interior of the polyhedron that converges to the optimal solution. Although the worst-case complexity of interior-point algorithms is much better than the worst-case complexity of the simplex algorithm, the latter is more efficient in practice.

Recently linear constraint solvers have been developed especially for use in interactive applications. Linear constraint solvers like QOCA [91, 90] and Cassowary [18, 7] are capable of solving hierarchies of linear constraints. QOCA's objective function minimizes the squared distances between the solution values and user-defined desired values. The Cassowary solver minimizes the weighted sum over the constraint errors of non-required constraints. Each constraint error is weighted by the constraint's priority. Both solvers use an incremental simplex algorithm that can update the optimal solution, instead of recomputing it from the start, after constraints have been added or removed. Hiroshi Hosobe has developed a number of powerful constraint solvers for systems of linear and nonlinear constraints that also support constraint hierarchies [59, 60, 61].

The availability of many industry-strength solvers favors the use of linear constraints: CPLEX is one of the leading commercial solvers and it can be used for Linear Programming and Mixed Integer Programming problems [74]. Two well-known open-source solvers are CLP [25] and DyLP [26] both of which implement different versions of the Simplex algorithm. Another commercial solver, Mosek [95], solves problems combining linear and conic equations of the following forms:

1. Quadratic cone:

$$x_1 \geq \sqrt{\sum_{j=2}^{m} x_j^2}, \qquad \forall i : x_i \in \mathbb{R}$$

2. Rotated quadratic cone:

$$2x_1x_2 \geq \sum_{j=3}^{m} x_j^2, \qquad \forall i : x_i \in \mathbb{R}.$$

Such conic constraints have been used in some layout applications that will be presented later in this chapter. All of these solvers support the OSI standard programming interface [27] and can be exchanged without requiring any code changes.

2.1.4 Geometric and Non-Linear Constraint Solving

Some CAD applications let the user draw a rough sketch of points, lines, circle segments, and arcs which have to be topologically correct. The user can add constraints such as fixed distances and angles, parallelism, incidence, perpendicularity, tangency, concentricity, and collinearity. The geometric constraint solver transforms the sketch into a precise drawing. Several constraint solvers exist that solve such systems of geometric constraints [21, 40, 83, 41, 60].

Non-linear numerical constraints such as quadratic equations have rarely been used. Iterative methods like Newton-Raphson solve such constraint systems numerically starting from a user-supplied initial solution [100, 56]. Some authors have researched the possibility of approximating non-linear constraints with linear equations [70, 93]. Recently, an algorithm solving systems of hierarchical non-linear constraints has been presented, although the algorithm is not guaranteed to find globally optimal solutions [61].

2.2 Interactive Applications

Following in the footsteps of Ivan Sutherland, many researchers built on SketchPad [113] and explored better user interfaces and different constraint satisfaction algorithms.

2.2.1 Constraint-Based Drawing

One of the earliest direct manipulation drawing programs using constraints was Juno [100] later followed by Juno-2 [56] both of which shared important characteristics. Juno was a *what you see is what you get* (WYSIWYG) graphical editor allowing the user to compose drawings of points and lines. Four constraints were supported by Juno: The distance between two pairs of points could be made equal, the vectors between the points could be made

parallel, and a vector between two points could be set to a vertical or horizontal orientation. Juno used an iterative numerical algorithm that converged towards the optimum starting from an approximate solution. The user had to draw a roughly correct sketch that served as the solver's starting point.

Juno and Juno-2 provided a second interface to the same drawing via its imperative language. Both, the WYSIWYG view and the programming view were shown in parallel as in Fig. 2.3. Each interactive change to a drawing resulted in a change of the code. A proficient user could edit the code, e.g., to create new constraints from the existing primitives.

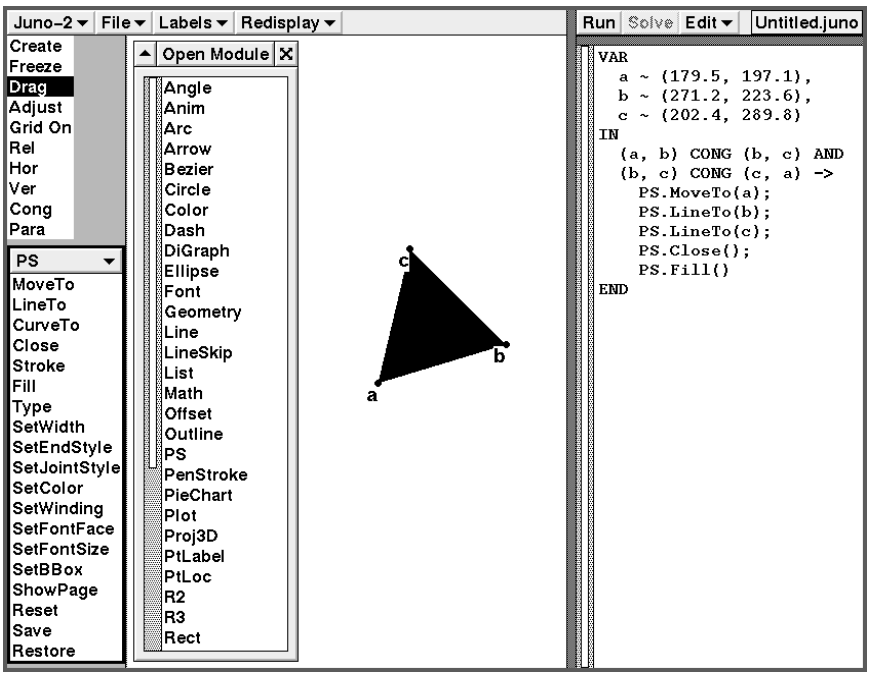


Figure 2.3: Juno-2's dual view interface showing both the drawing and the code producing the drawing side-by-side. Image originally appeared in [56].

GRACE [2] and Rockit [79] both tried to infer constraints directly from user interactions. GRACE was a simple prototype for drawing points and lines. Constraints could be added both by the click of a button and subsequent selection of the participating points, or by *demonstrating* the desired relation. When a point was dragged, it could be snapped to another point so both aligned horizontally or vertically. If GRACE was in demonstration mode, both points were aligned permanently, i.e., GRACE added a permanent alignment constraint. A point's position could be fixated by a long click on the point. Constraints such as fixed line length and slope could not

be specified by demonstration. While one object was dragged, the one-way constraint system was continuously solved to move constrained objects along.

Rockit provided no explicit mechanism to specify constraints. Every constraint was inferred. When an object was dragged close to another one, the constraint inference mechanism was invoked. Depending on the relative positions of the dragged and the stationary object, Rockit offered the user a choice of possible constraints ordered by probability. Rockit supported six constraint types: point coincidence, fixed distance, maximum distance, minimum distance, containment, and alignment. The inference mechanism was not very specific and moving an object close to another could imply at least a fixed distance, maximum distance, or minimum distance constraint. By displaying a list of all inferred relations, Rockit gave the user ultimate control over the establishment of constraints. By default, all constraints were undirected although the user could make individual constraints directed. As in GRACE, the constraint system was solved during user interactions and one-way constraint systems were sufficient for that purpose.

The interactive specification of constraints was perfected by Michael Gleicher with Briar [44, 45, 46, 47], a powerful drawing program. Briar featured an interaction mechanism termed *augmented snapping*: Whenever the user dragged an object, e.g., a point onto a line or a circle, this relation could be made permanent by establishing a constraint. Briar introduced auxiliary objects that were not visible parts of the drawing but enabled the construction of complex constraints. With a fixed diameter circle as a helper object, two points could be constrained to have a fixed distance. The circle center coincided with the first point and the second point coincided with the circle outline. Using helper objects for the construction of geometric constraints led to undesirable consequences. It was difficult to find a distinct visualization for every constraint and with the large number of helper objects the display became cluttered quickly. This made a drawing hard to understand. Briar used numerical methods similar to Juno and it also used the user-defined drawing as a starting point. Because the constraint system was continuously solved during user interactions, any user-defined configuration was a feasible solution to the constraint system. Briar could be used to animate mechanical systems, fulfilling Ivan Sutherland's vision. Gleicher built a working model of a simple engine, shown in Fig. 2.4, that transformed the up-and-down motion of combustion cylinders to the rotating motion of a camshaft. After developing Briar, he applied constraint solving methods successfully to the field of computer animation [49, 48].

Ultimately, creating geometrical drawings with the help of constraints never became mainstream. Simpler approaches proved to be sufficient, except for certain niches: The latest versions (in early 2009) of state-of-the-art

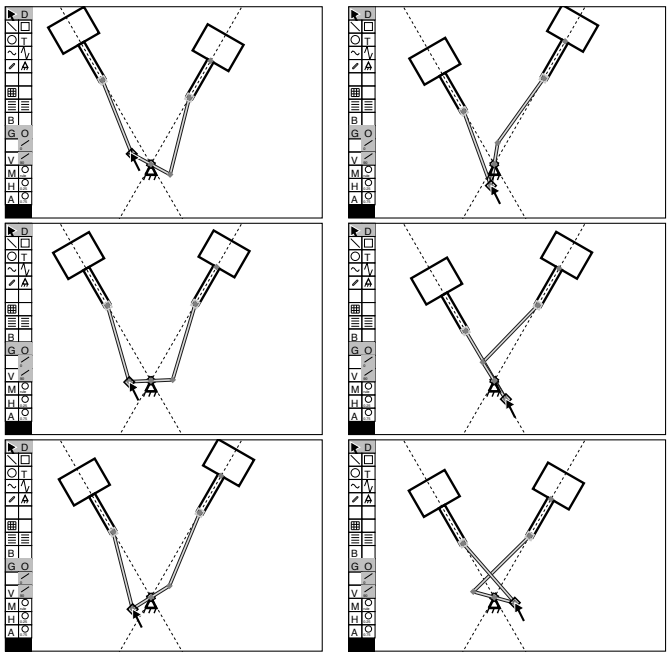


Figure 2.4: Model of an engine constructed with Briar. While the point is dragged, the constraint system is solved repeatedly and the engine is animated. Taken from [44]

CAD applications like TurboCAD, SolidWorks, and AutoCAD Inventor support constraints to model and simulate the behavior of mechanical systems. AutoCAD can be extended, e.g., with the IDX Variable Constraint System, to support constraints as well.

Constraints were also used in diagram editing applications. Helm et al. extended the UniDRAW framework, developed to facilitate the creation of custom graphical editors, and created a constraint-based diagram editor [55]. A diagram is a drawing composed of rectangular objects, some of which may contain text, that can be connected by arrows. In constrast to earlier constraint-based editors, the application used the QOCA linear programming solver [90]. The UniDRAW diagram editor used constraints to specify relations between shapes, such as alignment, and relations inside a shape, e.g., that a rectangle be square. The constraint system had to be solved continuously during user interactions. The user could influence the diagram layout by inserting spring constraints between shapes. The application calculated a minimum energy configuration over all springs and it minimized the distance each graphical object was moved from the user-defined position.

Spring constraints are still used, e.g., by the Apple User Interface Builder, to define the behavior of resizable dialog windows. The Size Inspector shown in Fig. 2.5 specifies an element's size and position relative to its container. In the *Size & Position* panel, the position of the element's corner or its center can be fixed. The *Autosizing* panel defines the element's behavior if its parent view is resized. *Struts* fix the distance between an element edge and the edge of its containing view. *Springs* define that the element resizes proportionally with its parent view along the specified direction.

Common drawing applications like CorelDraw, Adobe InDesign, Microsoft PowerPoint, or Microsoft Visio do not use constraints. At most, drawing programs allow the creation of guidelines to which objects can be attached. Moving a guideline then moves the attached objects [117]. A guideline can be represented by a simple one-way constraint.

2.2.2 Snap-dragging

The pragmatic *snap-dragging* approach, presented by Eric Bier in 1986 [13], has become a standard interaction technique. With snap-dragging, objects that are moved, rotated, or resized stick to preferred positions, angles, or sizes within a certain snap tolerance. When dragging a two-dimensional object in the plane, the edges of the moved object can snap to the edge of another object once the distance between both edges is less than the snap tolerance. When objects are rotated, certain angles like 0°, 45°, 90°etc. are usually preferred and the rotation angle can snap to them. Similarly, while

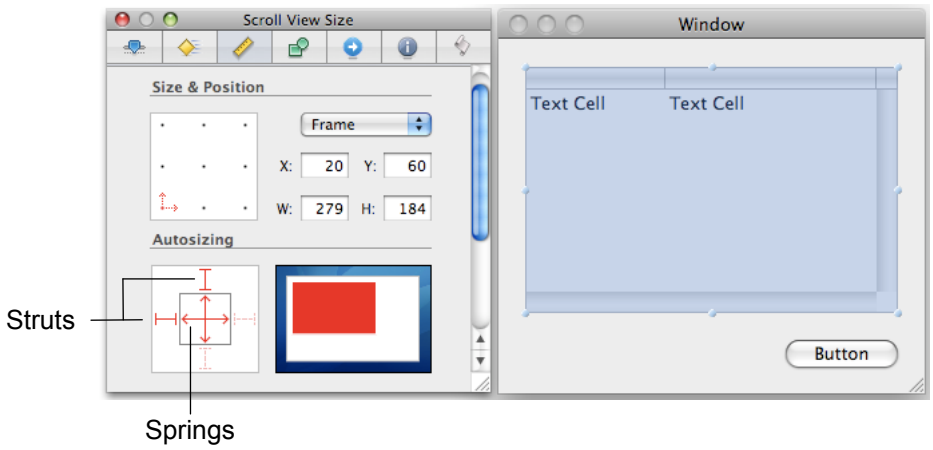


Figure 2.5: Spring constraints are used in the Apple Interface Builder to specify the dynamic behavior of user interface elements.

an object is scaled, its size may snap to multiples of its original size or to multiples of some other unit. With snap-dragging, users can interactively create precise drawings.

Snap-dragging constrains the user interaction but the constraints are not made permanent. Non-permanent constraints have also been used in automatic beautifiers. Pegasus [73] lets the user sketch a drawing and infers geometric constraints from the vectorized strokes. Inferred constraints are enforced a single time but are not made permament. The inferred constraints aid in the construction of precise line-based illustrations by making lines automatically perpendicular, parallel, or the same length.

2.2.3 User Interface Toolkits

Parallel to the development of constraint-based graphical editors, several research groups devised user interface toolkits that made the development of such applications easier. The first such toolkit was ThingLab [20, 16], an extensible kit-building kit developed in SmallTalk. Users could implement new SmallTalk classes, giving them both a visual representation and behavior. Class instances could be combined interactively in ThingLab's graphical user interface to form dynamic systems.

Peridot [97] and its successors Garnet [98] and Amulet [99, 96] were focused on the construction of interactive graphical applications. Peridot was a visual programming environment in which user interface components could be arranged interactively. Peridot featured a rule-based inference engine that

helped users specify the application behavior. Garnet provided much more complex user interface gadgets, added a programming interface, and included *interactor* objects. Interactors implemented common user interface actions, e.g., dragging, inserting text, or navigating menus. Amulet was reduced to a rapid application development framework. The Amulet framework already supported functionalities like undo, copy and paste, object selection and resizing via standard interfaces. New graphical widgets only had to implement the interface functions. Similar UI toolkits that enabled the user to specify interface behavior with constraints included Rendezvous [57] and the Penguims system [66, 65].

2.2.4 Summary

Ben Shneiderman coined the term *direct manipulation interfaces* for interactive systems that give the user direct control of objects via physical actions [109, 110]. Shneiderman identified key properties of highly usable human-computer interfaces. His analysis provides the necessary criteria to evaluate the achievements and shortcomings of constraint-drawing interfaces. The three principles of direct manipulation interfaces are

1. Continuous representation of the objects and actions of interest;
2. Physical actions or presses of labeled buttons instead of complex syntax;
3. Rapid incremental reversible operations whose effect on the object of interest is immediately visible [110].

Not every graphical user interface fulfills these requirements and, conversely, direct manipulation interfaces do not have to be graphical interfaces. Window managers on all operating systems allow the user to manipulate the position and size of application windows interactively. Until a few years ago, the window content did not update while the user dragged a window. The manipulated object was thus not continuously represented, only its outline would have followed the mouse cursor. Direct manipulation systems should give the user the feeling of interacting with objects themselves, and continuously representing the objects on the screen is important in maintaining this illusion.

Shneiderman suggests that application interfaces that incorporate these principles have many beneficial attributes:

1. Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.

2. Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features, because they can explore areas of interest interactively and do not have to learn a complex syntax.

3. Knowledgeable intermittent users can retain operational concepts.

4. Error messages are rarely needed.

5. Users can see immediately if their actions are furthering their goals, and if not, they can simply change the direction of their activity.

6. Users have reduced anxiety because the system is comprehensible and because actions are so easily reversible. (See [109], p. 251)

Hutchins, Hollan, and Norman [72] analyze how direct manipulation interfaces can reduce the user's cognitive effort both in performing a task and in understanding the presented result. Direct manipulation interfaces are supposed to reduce the distance between the task the user has in mind and the way the task can be achieved using an application interface. Hutchins et al. call this the *gulf of execution*. Direct manipulation interfaces also have to minimize the distance between a system's output and the user's understanding, termed the *gulf of evaluation* by the authors. Results should be presented in a way that corresponds with the user's mental model so she can easily perceive, interpret and evaluate them.

Adapting a user interface to the mental concepts of its users reduces the application's generality as Hutchins et al. describe. Common tasks can be made extremely simple, even for novice users, but tasks other than those foreseen become impossible to accomplish. This lack of generality is one of the most important disadvantages of direct manipulation interfaces that the authors mention.

The constraint-based drawing applications presented in this chapter were general-purpose drawing applications, supporting typical geometric primitives like points, lines, and rectangles. Their interfaces implemented many of the principles of direct manipulation outlined above. Juno and Juno-2 extended the graphical interface with a constraint equation editor. The powerful new concept of constraints they introduced, significantly widened the gulf of execution and the gulf of evaluation.

Expressing Intentions Using Constraints

The user has to understand how to express his intentions in terms of constraints, i.e., she has to bridge the gulf of execution. In very few application

scenarios do users actually think in terms of constraints. Engineers who use CAD applications to simulate mechanical systems, to engineer car engines, or to analyze the structural properties of a building understand what a constraint is and can think in terms of constraints. The user proficient in using a document processor and a presentation program who wishes to draw a flow chart almost certainly does not know what a constraint is.

Many of the presented applications used constraints to create new graphical objects from geometrical primitives by connecting lines, fixing their length, and constraining their angles. Most interactive applications today achieve the same goal by *grouping* several objects, effectively creating a new, compound object in the process. The object group behaves like a single object and can be dragged, rotated and scaled. Constraints have also been used to declare that two objects in a diagram be connected by a line. A connecting line has several important properties: It should not intersect other objects, it should be rectilinear, and have as few bends as possible. It is difficult to express these requirements as generic constraints. A procedural algorithm, on the other hand, can recompute optimal routings of connecting lines on-the-fly while an attached object is dragged.

To a user drawing a flow chart, the concept of grouping objects and of drawing a rectilinear connecting line are immediately obvious. These concepts correspond to the user's needs in a way that generic constraints do not. Constraints could be seen as a low-level technique that could be employed to maintain desired properties like the grouping of primitives. But exposing them to the user demands too much prior knowledge that most users do not possess.

Understanding Constrained Drawings

Constraints can affect the behavior of drawn objects in ways difficult to predict. Constrained drawings thus widen the gulf of evaluation. In the presented applications, constraints were used to define object behavior. If a single object was selected by the user and subsequently moved, other objects were moved too if they were connected by a chain of constraints. In studies, users have reported that such a behavior is highly unintuitive [117]. Although the user has manipulated only a single object, the effects of his action propagate through the whole drawing. At best, the cognitive effort required to predict such effects is reduced by visualizing every constraint as Briar did. This approach increased the visual clutter and made drawings hard to understand in another way.

All presented constraint-drawing applications restricted user interactions. Briar, e.g., enforced all constraints during a user action. A user could not

perform an action that was prohibited by a constraint, unless the user deleted the constraint explicitly. This implies very strong constraint semantics. A constraint expresses a permanent property of the system that must be upheld even if further user actions contradict it. In a CAD application, the user may specify the thickness and length of a steel bar spanning a building floor. Such a constraint must restrict the permitted user actions. The constraint prevents the user from widening the space that the steel bar spans beyond the maximum length at which the bar can still support the weight of the next floor. Such strong constraint semantics require the user to think in terms of constraints, i.e., the user must consider the constraints an integral part of the drawing.

The constraint solver adds another level of complexity. Both the user and the constraint solver may change the positions and sizes of objects. If the user understands the constrained drawing in front of him and knows how to express his intentions in terms of constraints, there is still an element of uncertainty because the constraint solver may not calculate the intended result. In the presented applications, the user specified all object positions and sizes. After the user had added some constraints, the constraint solver tried to recompute both the object positions and sizes. In order to maximize the predictability, all systems minimized the deviation from the user-specified input. In many situations, the results are close enough to the user's input so the user is not surprised. A small change could make the previous solution infeasible, causing the constraint solver to make significant and unexpected changes.

Conclusion

Although many researchers explored the possibilities of using constraints in interactive drawing applications, they have not been widely adopted in mainstream drawing applications. The capabilities of constraint-based applications only outweigh their additional complexity for users that are able to express their intentions in terms of constraints.

2.3 Document Layout

The research area of automatic document and page layout is both active and successful. Almost everyone working with computers will have had some contact with automatic document layout techniques. Interactive word processors perform basic layout tasks like line breaking, or the placement of floating figures. Every web browser renders tables depending on the size of the content

each table cell holds. Document layout systems like LaTeX solve all of these problems with little user intervention. In the following, the term document layout is used both for multi-page and single-page layout problems.

2.3.1 Task-specific Layout Algorithms

The simplest content arrangement is a regular grid. In a grid layout, the document elements are fitted to an integral number of cells. Feiner's GRIDS system [36] generated an optimal grid from the document content, display properties, and user preferences. The layout algorithm determined the optimal size of a grid cell depending on the distance at which the information was read, basic rules of legibility, and the amount of text contained in each cell. This information was expressed using a special display grammar.

Graf's LayLab application followed a similar approach [52]. It generated the grid automatically and matched the document content to the individual grid cells. It combined two different constraint solvers, a numerical local propagation solver and a finite-domain solver. The interactive visual programming tool InLay [51], used to draw and layout flow charts, and the automatic yellow page pagination tool YPPS [50] were both based on LayLab.

The yellow page layout problem has been solved with simulated annealing [77], combinatorial algorithms [102], as well as constraint-based approaches [50]. The precise problem definition made it an attractive target for layout algorithms. Yellow page layout is a combination of two bin-packing problems. A stream of text entries has to be arranged in multiple columns and a stream of larger advertisements, each belonging to a text entry, has to be placed on the page. Both streams have no other semantical relations.

The table layout problem is similarly suitable for automatic layout algorithms. A table cell can span several rows and columns and each cell has constraints on its width and height. Stated as a decision problem, the question is to decide whether the table fits into a rectangle of given width and height. Most algorithms solve the optimization problem of finding a table with minimum height for a given width. If a cell contains text there is a discrete set of minimal width-height-configurations that are large enough for the text to fit in. In that case, the table layout problem is an NP-complete combinatorial optimization problem [115]. Efficient algorithms exist to enumerate each cell's minimum height width-height-configurations if the text has uniform height [3]. The width-height-configurations can be approximated using linear [84], or conic constraints [71, 69]. Iterative algorithms calculate sequences of table layouts with decreasing height, starting from an initial solution with minimum width [71].

The problem of placing floating figures in documents containing text and images [92, 68] has recently been investigated. Current document processors like Microsoft Word or LaTeX follow a first-fit approach. The floating figure is placed at the first position behind its insertion location where it fits. Nathan Hurst and Kim Marriott search a *layout without obvious errors* or LWOE. They explain that humans rarely try to achieve a globally optimal layout. Instead, an initial draft solution is optimized until it cannot be improved by making obvious local changes. The proposed algorithms follow this iterative approach to quickly find locally optimal solutions satisfying aesthetic criteria.

2.3.2 Document Template Algorithms

The layout algorithms explained so far cannot be used to layout rich documents containing text, illustrations and floating figures. Each algorithm solves a special layout subproblem. Several researchers have tried to calculate general document layouts by using adaptive templates. Adaptive templates express document layouts with constraint equations instead of fixed values for positions and sizes. The Scalable Vector Graphics format (SVG) has been extended to support constraints [8]. An adaptive template can be limited to the output medium and a range of display sizes for which it has been optimized. Such templates were developed by Jacobs et al. [76, 75] for the rich layouts of magazine pages containing sidebars, overlapping images, and floating figures. Documents may contain sections with alternative content. The presented layout engine chooses the best template considering the output device and the document content and it decides which content is displayed depending on the available page space. Figure 2.6 shows the same document rendered for different page sizes.

Jacobs et al. presented an editing prototype for adaptive templates and for documents containing sections of alternative content. McCormack et al. developed an authoring tool for adaptive diagrams that supports the creation of different layout configurations, alternative content, and interactive behavior [94]. The authors have chosen multi-way constraints instead of linear arithmetic constraints. For each interaction, the changed input variable is known at compile time and the multi-way propagation algorithm can be compiled into code. The supported layouts are “grid-like”, i.e., all objects are placed with respect to some horizontal and vertical guidelines. The application environment chooses the optimal diagram layout depending on the display environment, user interaction, and user language.

Such template-based mechanisms generate documents with different content but the same layout. They solve document layout problems but they cannot calculate new layouts themselves. This task is delegated to the tem-

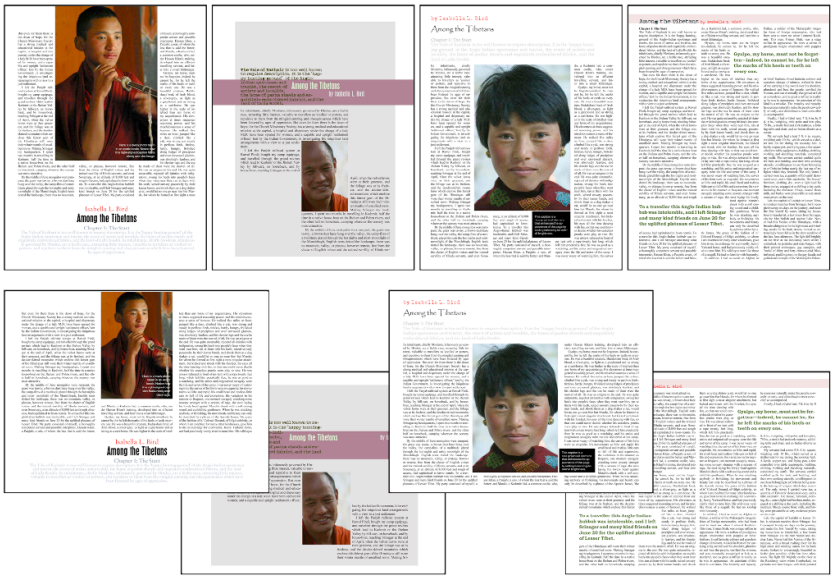


Figure 2.6: Examples of adaptive documents rendered using templates optimized for the different page sizes. Originally published in [76].

plate designer. The TALL template language [32] lets the user describe the high-level topological structure of the document. A TALL document structures the content by recursively defining groups of content elements. For each group, a desired layout topology can be chosen from a finite set, e.g., horizontal sequence, vertical sequence, columns, or circular arrangements. Alternatively, the topology can be left unspecified leaving the task of finding a good arrangement to the layout system. The layout algorithm calculates the final document layout bottom up, starting with the individual groups and working its way up to the final document. The layout engine calculates PDF documents from documents in TALL language although the authors note that an interactive authoring environment is planned.

2.3.3 Knowledge-based Document Layout

An algorithm can deduce a layout from the document content if it understands the document's semantics. Many visual languages have well-defined semantics. Layout algorithms for logic diagrams [111] and data flow charts [9] have been developed 20-30 years ago. The Penguins system was an authoring tool for diagram editors [24]. It allowed a designer to define the graphical constraints of a visual language and the Penguins system generated an inter-

active graphical editor for the specified language. Today, the Unified Modeling Language (UML) has become the de facto standard to visualize software designs. UML defines both the semantics and the visual representation of software components. Several research groups have explored automatic UML diagram layout algorithms [35, 34, 53].

If meaning cannot be derived from the content itself, it can be provided explicitly. In the last decades, several researchers have developed relational grammars describing presentation content [89, 88, 14, 116, 108]. A layout system interpreting such input is not bound to a specific visual representation. This idea has been applied to the creation of program user interfaces. SUPPLE creates user interfaces from a description of their function and information about the available user interface components [42]. The system is also capable of taking typical use-case scenarios into account, minimizing the time it takes the user to complete common tasks.

2.3.4 Random Algorithms

Only a few researchers have tried to solve document layout problems with randomized algorithms. Purvis treated the generation of aesthetic documents as an optimization problem [104] and solved it using genetic programming. This approach proved to be difficult. For the solutions to evolve towards better layouts, their genomes had to encode generic properties of aesthetic layouts. Several authors try to enumerate important properties of good layouts [54, 85], but the resulting parameters are very general. Given only measures of visual balance or aligned-ness it is very hard to evolve towards better solutions.

GADGET [38] implemented a generic framework for optimizing user interfaces with simulated annealing. The framework provided callback interfaces that let the developer implement evaluators and iterators. Evaluators calculate a quality measure for the current layout and iterators generate new solutions.

2.3.5 Specifying Complex Layouts

None of the applications presented so far could generate a variety of different layouts. The algorithms were either limited to one type of output or they required a certain kind of input document whose semantics could be analyzed. A more capable layout system would need an input language able to express diverse constraints on the document's appearance, and a layout algorithm capable of handling such input.

The Auckland Layout Manager, a user interface layout system, introduced the concept of *tab-stops* [87, 86]. Every user interface element is attached to tab-stops in horizontal and vertical direction, i.e., one on each side for a rectangular object. Tab-stops are partially ordered and the programmer may specify linear constraints on the positions of several tab-stops. A rectangle could, e.g., be constrained to a preferred size or a desired aspect ratio. A tab-stop position can be fixed in absolute coordinates, or its position may be specified relative to another tab-stop. User-specified constraints are internally represented as soft constraints, i.e., the constraints may be left unsatisfied if necessary. The constraint system cannot become infeasible. Instead, the penalties associated with unsatisfied constraints are minimized. The programmer can specify piece-wise linear penalty functions, i.e., the infeasibility penalty can depend on the constraint error itself. The authors do not go into detail how exactly the constraint system is solved. It seems an equidistant distribution of all tab-stops is taken as an initial guess which is then adapted to satisfy all constraints.

The Active Layout Engine [84] adapts existing document layouts for changed content, a process called Variable Data Printing (VDP). An existing document can be augmented with a description of important layout constraints, creating an Active Template. The original document, together with the Active Template and the new content are then fed into the Active Layout Engine (ALE). As in the Auckland Layout Manager, the document layout constraints can be arbitrary linear (in-)equalities, solved by the Cassowary linear constraint solver. The original document is used as the design blueprint and the layout algorithm tries to maintain its layout as well as possible. ALE approximates the discrete set of text box sizes with linear equations. At first, the set of text box sizes is approximated using a hyperbolic function which in turn is sampled with linear constraints. The authors describe a two-pass algorithm to compensate for the inaccuracies of the linear approximation: In the first phase the text box size is calculated under the described set of linear constraints. In the second pass, the text box size is constrained to the minimum size closest to the approximate solution. The authors' evaluation shows that the results of this two-pass method are close to the optimal solution.

Both systems can handle a wide variety of layouts because they impose few limitations on the acceptable inputs. ALM and ALE use linear constraint solvers instead of the previously common local propagation algorithms. The Auckland Layout Manager burdens the developer with the large amount of parameters that have to be set manually, which compared to other user interface toolkits is quite tedious. The Active Layout Engine requires the user to explicitly specify the constraints the finished document should satisfy.

The generation of these Active Templates is deferred to a yet unspecified upstream component.

2.3.6 Summary

If the term *layout* is loosely defined as the assignment of positions and sizes to graphical elements, only the knowledge-based layout and the table layout algorithms truly calculate a layout. Unfortunately, neither class of layout algorithms can be easily generalized to new kinds of inputs.

Many researchers have been concerned with template-based layout algorithms, a term that is misleading because the actual layout template has to be crafted manually. The algorithms only choose the best template for the given input. More complex applications like ALE and ALM take as input a user-created layout together with altered content and the user-created layout is adapted to the new content. ALE and ALM minimize the deviation from the user input similar to the approaches developed for constraint-based drawing applications.

Without the ability to deduce semantical relations between graphical elements, some user input is necessary to guide a layout algorithm. In fact, even if an algorithm completely understood a document – and thus was able to layout its content automatically – many users would still like some level of control over the resulting layout. What forms could the user input take? What is the smallest amount of user input imaginable?

At one extreme are the presented template-based layout algorithms which receive almost pixel-perfect manually produced page layouts as input. Given new content, the algorithms minimize the deviation from the template. As a side effect, every aesthetic property of the user-created layout – absolute positions and sizes, relative element positions, alignments – informs the calculation of the new layout even if the layout algorithms has no notion of those concepts.

If the information about absolute positions is removed from a layout template, it still specifies the total order of all layout elements. If the layout is compact, i.e., if there is no free space between elements, a layout algorithm only has calculate the element sizes from their content. Table layout algorithms solve this problem for rectilinearly delimited layout elements.

The information contained in the user input can be further reduced if the input only specifies a partial order over all page elements in every dimension. The ALM system can handle such inputs although its layout algorithm is very simple. If the page elements are only partially ordered, the problem is very likely under-constrained and the layout algorithm has to determine the best element arrangement as well as every element's position and size.

Finally, if all information about the spatial arrangement of the content is removed, the layout algorithm is left only with the content to analyze. Many layout problems such as graph layout and UML layout fall into this category. Graphs have little semantics but are only composed of two objects: nodes and edges. Thus, a graph layout algorithm can analyze its input to identify components, node clusters, or hierarchies of nodes connected by directed edges.

Thus, in an interactive layout application the user must at least specify a partial order over the page elements in both dimensions. Lutteroth et al. have already described an abstract representation of such a layout problem [87, 86]. However, it is still unclear how a user can specify such a layout problem interactively. As outlined in section 2.2.4 of this chapter, interactive applications must give their users the feeling of control. In a layout application, many decisions will be made not by the user but by the layout algorithm. It is thus a big challenge to design a user interface that still lets the user feel in control although in many ways he is not.

2.4 Interface Requirements of an Interactive Layout Application

In a layout application, the specification of constraints is mandatory. Every user action will specify a layout constraint. Still, the interaction must feel as natural and intuitive as in a common drawing application. The relation between a user's action and the visual result is only indirect however: The user input is fed to the layout constraint system whose solution is presented on the screen. Thus, the design of an interactive layout application poses many new challenges. Since the principles of direct manipulation interfaces are formulated in very general terms, it is necessary to analyze how they apply to the problem of designing an interactive layout application.

Understandability

A direct manipulation layout application must let the user interact directly with shapes and constraints. Therefore constraints must be visualized and the constraint semantics must be immediately obvious from the visual representation. Although constraint visualizations are necessary, they introduce visual clutter and hence additional complexity as experienced in Briar. This effect can be reduced if the number of available constraints is reduced to the minimum, i.e., only to constraints that represent very simple and familiar concepts in the user's mind. The number of constraints visible at the same

time can also be reduced, e.g., by only showing the constraints affecting the currently selected set of graphical elements.

During a user action, the manipulated objects must be permanently represented on the screen. If the effects of an ongoing user action are continuously visualized, the constraint solver has to run repeatedly during a user action. Obviously, this affects the system performance. But too much visualization during a user action can increase the user's confusion rather than his comprehension. While the user manipulates an object, e.g., while he moves it over the page, the object goes through many intermediate states between its original position and the desired target position. If the user has at least a rough idea where the element should be placed based on his own preferences, the intermediate visualizations will only surprise and detract him or her but they will not provide any important information.

Reversibility

If the system gives some visual feedback during a user interaction, the user can reevaluate and cancel the action at any time. The system must support undo and redo with an infinite number of undoable steps so that any action that – despite the visualization – did not have the intended result can be reverted with the click of a button.

Reproducibility

The same sequence of user actions must always yield the same result, i.e., the system behavior must be reproducible. This implies that the system must have no hidden state that the user is unaware of and that cannot be directly manipulated. The requirement of reproducible results limits the applicability of random algorithms in interactive applications.

In previous constraint-based drawing applications, the user could make drawings without any constraints. Constraints were used to improve the accuracy of a drawing, but they were not required. It was thus a natural idea to change the initial drawing minimally until all constraints were satisfied, but this made it difficult to reproduce the same drawing.

A layout application, on the other hand, is expected to tolerate large deviations in the user's input and – within certain bounds – should arrive at exactly the same result. If the exact input positions influenced the layout process, the user would have to put great care into his initial drawing which would run counter to the whole idea of an automatic layout system. Thus, a layout algorithm must calculate the same layout reproducibly even given altered input.

Controllability

No matter how good the results of a layout algorithm are, some users will have very specific ideas how the page content should be laid out. A layout system has to offer its users the power to specify exactly where an element is placed and what size it has. Such user decisions should be final, i.e., they must be respected by the layout algorithm. The user should be able to place a few elements at exact positions while the remaining elements are positioned by the layout algorithm. Thus, if necessary, the user has full control over the layout.

Previous constraint drawing applications maintained all constraints during user interactions. Changing some property of a single element could have caused several other objects to change as well, if they were connected by a chain of constraints. Users frequently found this behavior hard to understand [117]. An action's consequences are much easier to understand if they remain local. Thus, every user manipulation should only affect the selected graphical elements and all other elements should be left unchanged. Constraints between the manipulated objects and the rest can be removed if they become unsatisfiable. Such an approach is feasible in layout applications but not, e.g., in CAD applications as explained in section 2.2.4. In a layout application, constraints are merely a way to persistently represent prior user decisions which the user can override and change at any time.

Over- and Underspecifiability

A layout system must be able to deal with under- and over-constrained inputs. Under-constrained inputs are the default for a layout algorithm. If the layout were completely specified, there would be no need for a layout algorithm. Over-constrained inputs will occur frequently, e.g., if the space required by the content on a page exceeds the available page space. For over-constrained inputs, the layout system must calculate a layout that violates as few constraints as possible. The solution should indicate visually which parts of the input are too constrained. Essentially, the layout algorithm must always generate a result and is never allowed to fail with an error message.

Chapter 3

System Overview

This thesis proposes novel techniques to layout a wide array of documents. The contributions of this work include both an algorithm to solve a generic class of layout problems and an intuitive interface to specify instances of such problems. The layout algorithm and the user interface have been implemented to demonstrate their usefulness. The resulting system is called the *Interactive Constraint-Based Slide Manipulation* system or *ICBM* in short. This chapter presents the problem ICBM is designed to solve and explains the basic architecture of the ICBM system.

3.1 The Chore of Manual Document Layout

Figure 3.1 shows a sequence of document layouts a user may have created with the following actions:

1. The user picks the desired chart type from a menu and clicks at the desired insertion position,

2. inserts the chart legend at the top-right corner of the chart, and

3. selects a text box from the menu which he resizes to align left and right with the chart. The result is shown in Fig. 3.1a.

4. The user inserts a narrow table to the right of the chart and starts typing. There is not enough space to fit all the necessary text.

5. The text box, the chart, and the legend have to be selected and moved to the left until the table can be made wide enough. This may take several tries until the resulting layout looks as depicted in Fig. 3.1b.

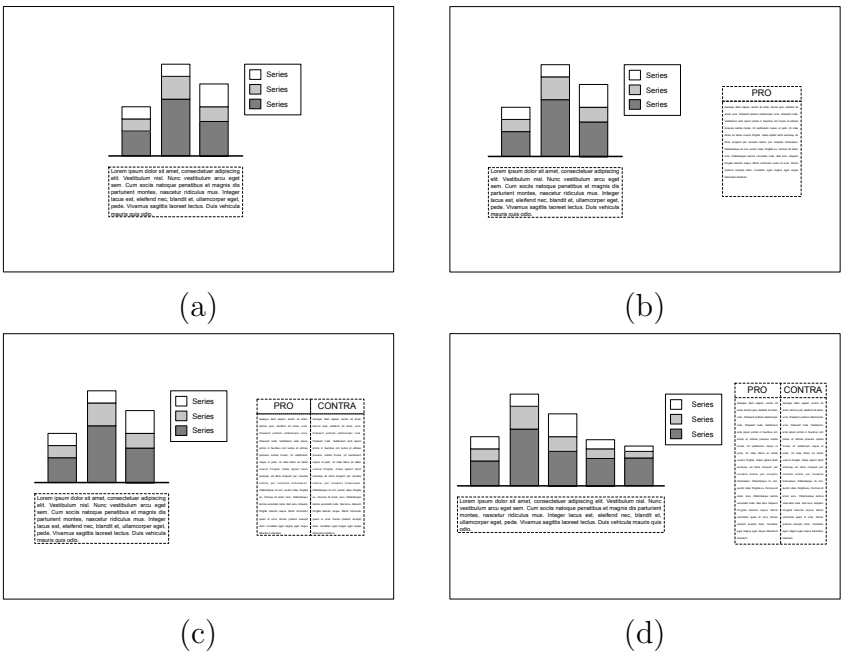


Figure 3.1: Adapting layouts when the content changes.

6. The user reflects on the information he would like to convey, decides to present both sides of the argument, and adds another table column.

7. Again, the text box, chart, and legend have to be moved to the left to accommodate the wider table.

8. The user has to make both table columns narrower and taller. Figure 3.1c shows the slide after this step.

9. Finally, the user's boss drops in to say that she would like to see the latest data included in the chart. The user thus adds two columns to the chart.

10. First, the user has to realign the left and right text box edges with the wider chart. The text box height can be reduced too.

11. Then the legend has to be moved next to the chart again.

12. The text box, chart, and legend have to be moved until they almost align with the left page border.

13. The user has to resize both table columns again, making them narrower and taller until the final layout of Fig. 3.1d is reached.

This example illustrates how much time is lost rearranging document elements after the content has been changed: Of the 13 actions necessary to create the final result, 7 are only needed because the layout does not adapt automatically to changed content. As the number of elements on the page increases, the more time is spent rearranging the elements each time the content is edited.

3.2 Interactive Layout of Presentation Slides

How could creating the same document be made easier? Under which conditions can a layout application make these decisions automatically? The example suggests a few possible solutions:

- If the alignment of the text box to the left and right chart edge was persistent, the text box would stay aligned to the chart if the latter was made wider by additional data. The same is true of the chart legend.
- If text boxes resized automatically to fit their content, the user would not have to adapt the table column sizes or the text box height.
- Finally, adjacent elements, like the chart and the table, could stay adjacent even if one of the elements increased in width. If the chart moved to the left automatically, instead of overlapping the table whenever either of them was made wider, the user would not have to fix the layout manually at all.

Modern desktop applications like Microsoft PowerPoint, Apple Keynote or CorelDraw have user interfaces that share many familiar well-established interaction techniques. They follow the principles of direct manipulation interfaces. The user interacts directly with graphical objects. The user can select the objects of interest with the mouse to resize, rotate, move, or delete them. These interfaces give the user total freedom. For general-purpose drawing programs like CorelDraw or Adobe Illustrator this is a necessity. Some users demand total control to produce results like those shown in Fig. 3.2[1].

At the other extreme are the users who delegate all control to a document layout system, e.g., LaTeX. Most documents have a simple structure. The order of chapters and paragraphs is defined by the writer, and floating

[1]Images copyright of Enjin Magazine (http://www.enjin.co.za), Turbulence Magazine (http://turbulence.org.uk), Wired (http://www.wired.com), and notion (http://www.notionmag.com).

Figure 3.2: There are classes of users demanding total freedom to express their creativity.

figures can be placed close to their reference in the text. The rules of good typography are old and well understood and, as the LaTeX system shows, they can be applied automatically [81].

Between those extremes of total control and total delegation lies the problem of laying out presentation slides. Presentation slides are created interactively and often contain complex arrangements of text and graphics. Every time the content is redacted, the slide layout has to be recreated. Yet, in business environments, little artistic freedom is allowed and regular, standardized slide layouts are preferred. Completely automating the generation of slide layouts is hard because the different building blocks lack well-defined semantics. Thus, creating presentation slide layouts could be a problem best solved by a user interacting with an automated layout system.

A presentation slide is composed of arbitrarily positioned graphical elements like rectangular text boxes, other shapes like box arrows and stars that may also contain text, charts, and pictures. Since slides are often used to support the argument of a human presenter, the best arrangement of elements on a slide may depend on the argument the presenter is trying to make. In fact, arranging the elements on a slide in the most logical order can be a big help in developing a convincing argument. Thus, there are important reasons why the layout of presentation slides is best done interactively.

3.3 The ICBM System

If the user has placed some objects on the slide, a layout algorithm has to calculate their positions and sizes while maintaining layout properties important to the user. There are three aspects to this problem. First, the user has to interact with the system, specifying the spatial arrangement of the slide objects, i.e., a partial order over the slide objects in two dimensions. Second, the layout thus created has to be interpreted and important layout features have to be identified that should be maintained. The third and final step is to calculate a new layout that maintains the object topology and distributes the available slide space among the slide objects and the inter-object space so that the overall layout appeals to the user.

3.3.1 User Interaction

The ICBM user interface follows the requirements detailed at the end of the previous chapter. It reuses the metaphors, interaction mechanisms, and visualizations of familiar programs like Microsoft PowerPoint: The user selects one of several graphical elements in a menu and inserts it by moving the mouse to the desired position and clicking the mouse button. One or multiple elements can be selected by clicking at them while keeping the Ctrl-key pressed. If an element is selected, manipulation handles appear at its corners.

The user interactions are internally transformed into a constraint system which is then solved to create a new layout. For example, if a user aligns the edges of two shapes, a persistent alignment constraint is created and the layout algorithm keeps both edges aligned until the user removes the constraint, e.g., by manually moving one of the edges. In addition to alignment constraints, the user can specify the absolute position of an element, its exact width or height, the distance between two element edges, and she can specify that several elements should have the same width or height. The choice of explicit constraints is very limited but the user has full control over the layout if desired by specifying the absolute element positions and sizes.

3.3.2 Implicit Constraints

In addition to the explicit constraints specified by the user, implicit constraints are added by the layout system itself. Implicit constraints maintain important features of the layout that the user does not expect to change. Text boxes are implicitly constrained to completely enclose their text content. The layout system tries to maintain the topology of the elements, i.e., if a table has been inserted to the left of a chart, the table and chart must

not switch sides. Figure 3.3 shows the implicit gap constraints as arrows. In this example, there is no constraint specifying the vertical position of the table in relation to the vertical position of the chart.

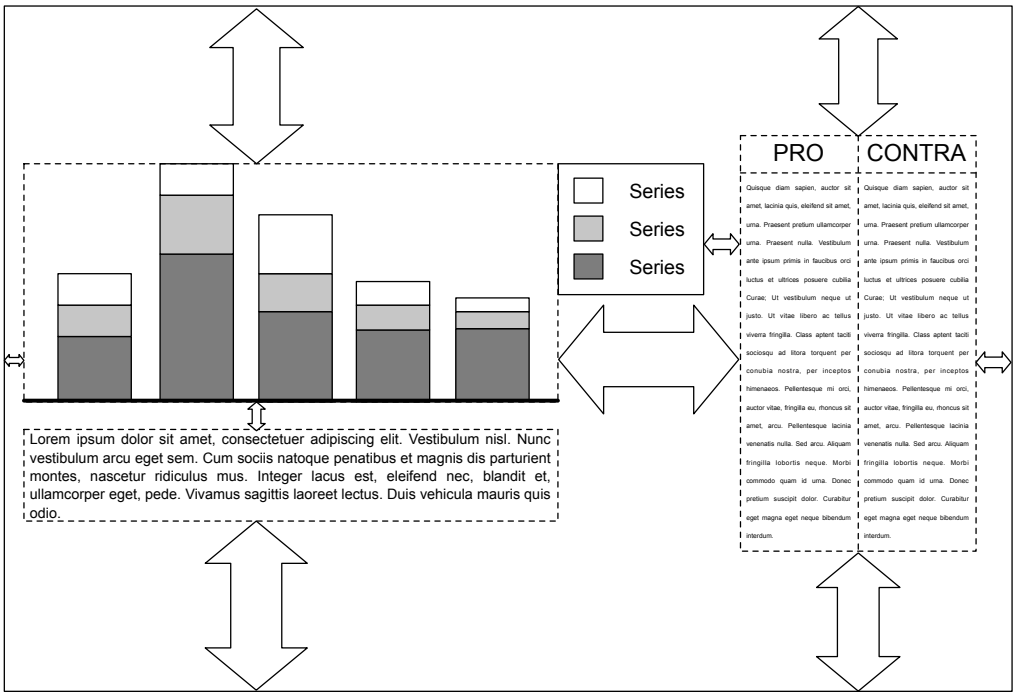


Figure 3.3: Implicit gap constraints to separate adjacent shapes and to distribute the available space.

3.3.3 Layout Algorithm

Given an abstract representation of the elements on a slide and the constraints between them, the layout algorithm finds a space distribution between the elements and the inter-element gap space. The user interface and the layout algorithm allow the creation of very different types of documents, some of which have previously required special-purpose layout algorithms. Users can create slides with complex tables, organizational charts, flow charts, logic diagrams, UML diagrams, all of which are automatically laid out.

3.3.4 Implementation

The ICBM layout system is built on top of the think-cell PowerPoint Addin Framework. think-cell chart is a charting software that integrates with PowerPoint, thus giving users an application environment they are familiar with, while at the same time integrating a powerful new user interface. ICBM uses the seamless integration with PowerPoint to ease the transition period for novice users. Compatibility with PowerPoint is of course a commercially important argument. The Addin Framework allows complete flexibility in developing the necessary constraint visualizations and the user interface as a whole. All algorithms described in this thesis are self-contained and do not have any dependencies on the Addin Framework or on the integration into PowerPoint. The system has been developed using C++ and the layout system alone comprises about 15000 lines of code. The layout algorithm described in Chapter 5 uses the CLP linear constraint solver [25].

3.4 Modeling the Application Domain

The ICBM user interface does not restrict user interactions in any way. The object model must be able to represent this diversity.

Definition 8. A **gridline** is defined by a pair $g = (v, f)$ where $v \in \mathbb{R}$ is the coordinate or *value* of the gridline in either horizontal or vertical direction and $f \in \{true, false\}$ is *true* if and only if the gridline position can be changed by the layout algorithm.

Less formally, a gridline represents a coordinate in either x- or y-direction.

Definition 9. A **gridline list** is a list of gridlines ordered by their value

$$G = (g_0, \ldots, g_i, g_{i+1}, \ldots, g_n) \qquad g_i.v \leq g_{i+1}.v.$$

Definition 10. A **shape** is a tuple $s = (G_x, G_y, span_x, span_y)$ where G_x is a horizontal gridline list and G_y is a vertical gridline list.

$$span_x : G_x \to \{(g_i.v, g_j.v, d) \mid g_i, g_j \in G_y, i < j, d \in \{low, interior, high\}\}$$

is a function that assigns each horizontal gridline in $g \in G_x$ an interval of y-coordinates where shape s occupies g and a direction d on which side g is occupied. $span_y$ is defined identically for gridlines $g' \in G_y$.

For convenience, the span of gridline g in direction d

$$span : \mathcal{G}_{x/y} \times \{low, interior, high\} \to I_{\mathbb{R}} := \{(a, b) \mid a, b \in \mathbb{R}\}$$

is defined as

$$span(g,d) = \bigcup_{s \in \mathcal{S}} \left\{ (g_i.v, g_j.v) \,|\, g \in s.G_{x/y} \wedge (g_i.v, g_j.v, d) = s.span_{x/y}(g) \right\}.$$

Correspondingly, the complete span of gridline g $span : \mathcal{G}_{x/y} \rightarrow I_{\mathbb{R}}$ is defined as

$$span(g) = span(g, low) \cup span(g, interior) \cup span(g, high).$$

Definition 11. A **page** is a triple $P = ((0_x, w_x), (0_y, h_y), \mathcal{S})$ where $(0_x, w_x)$ are the left- and right-most gridlines, fixed at values 0 and the page width w, and $(0_y, h_y)$ are the top- and bottom-most gridlines. $\mathcal{S}$ is the set of all shapes.

Set $\mathcal{G}_x(P)$ or short $\mathcal{G}_x$ is the global set of all horizontal gridlines, i.e., $\mathcal{G}_x = \bigcup_{s \in \mathcal{S}} s.G_x \cup \{0_x, w_x\}$. Correspondingly, $\mathcal{G}_y$ is the set of all vertical gridlines. Many algorithms that will be discussed in the next chapters receive a subset of gridlines as input that is either $G \subseteq \mathcal{G}_x$ or $G \subseteq \mathcal{G}_y$. In these cases where the exact orientation does not matter as long as all gridlines have the same orientation, the gridline sets will be denoted $G_{x/y}$ or $\mathcal{G}_{x/y}$ respectively. Gridlines resemble the concept of tab-stops used by Lutteroth et al. to represent positions in the Auckland Layout Manager [86].

The semantics of these objects is illustrated by the pentagon shown in Fig. 3.4. A pentagon is a shape with three gridlines in x- and y-direction.

$$\begin{aligned}
G_x &:= (g_x^0, g_x^1, g_x^2) \\
G_y &:= (g_y^0, g_y^1, g_y^2) \\
span_x(g_x^0) &:= (g_y^0.v, g_y^2.v, high) \\
span_x(g_x^1) &:= (g_y^0.v, g_y^2.v, interior) \\
span_x(g_x^2) &:= (g_y^1.v, g_y^1.v, low) \\
span_y(g_y^0) &:= (g_x^0.v, g_x^1.v, high) \\
span_y(g_y^1) &:= (g_x^0.v, g_x^2.v, interior) \\
span_y(g_y^2) &:= (g_x^0.v, g_x^1.v, low)
\end{aligned}$$

The definition of a shape via the attached gridlines and the occupied spans on those gridlines has two advantages. First, two shapes are aligned to each other if they share the same gridline in either x- or y-direction. Second, the definition is general enough to accommodate a huge variety of graphical objects and it contains only the minimum information necessary for

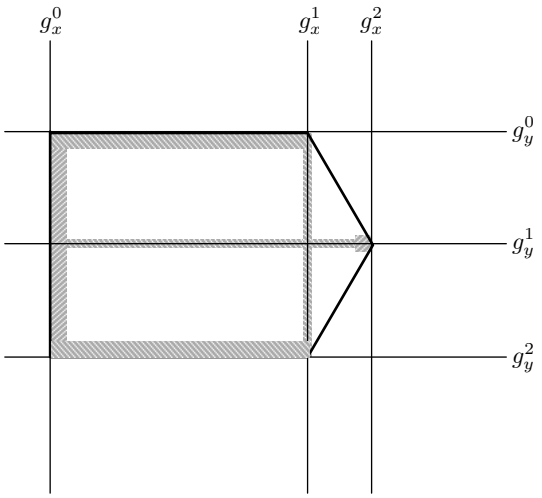


Figure 3.4: A pentagon is attached to three gridlines in each dimension with occupied spans as shown

the layout algorithm. Especially, all information on how a shape is actually drawn is removed. A simple text box and a complex column chart are both defined via two x-gridlines and two y-gridlines with spans covering every side. For the layout algorithm both objects are identically defined, although visually they are quite different.

If the above pentagon is the only shape on a page, it is easy to establish the set of constraints that have to be maintained:

$$
\begin{aligned}
0_x.v < g_x^0.v < g_x^1.v < g_x^2.v < w_x.v \\
0_y.v < g_y^0.v < g_y^1.v < g_y^2.v < h_y.v \\
g_y^1.v - g_y^0.v = g_y^2.v - g_y^1.v \\
g_x^2.v - g_x^1.v = 0,28 g_y^2.v - g_y^0.v
\end{aligned}
$$

The first two lines guarantee that the horizontal and vertical pentagon gridlines do not change their order and that the pentagon stays on the page. The third line maintains that gridline g_y^1 and thus the pentagon tip stay vertically centered and the last line guarantees the preferred ratio of pentagon height to arrow head length. The ICBM system supports five kinds of constraints.

- **Minimum Distance Constraints**

$$
C_{Min} := \left\{ \sum_{i=1}^{n} c_i \cdot g_i.v \geq d \,|\, \neg g_i.f, g_i \in \mathcal{G}_x \cup \mathcal{G}_y, c_i, d \in \mathbb{R} \right\}
$$

constrain a linear combination $\sum c_i \cdot g_i.v$ of unfixed gridline values to be greater or equal than some constant d. The value of fixed gridlines, such as the page boundaries, is never changed by the layout algorithm. If a fixed gridline is part of a constraint, its value can be accumulated in the constant part of the above equation.

- **Maximum Distance Constraints**

$$C_{Max} := \left\{ \sum_{i=1}^{n} c_i \cdot g_i.v \leq d \,|\, \neg g_i.f, g_i \in \mathcal{G}_x \cup \mathcal{G}_y, c_i, d \in \mathbb{R} \right\}$$

constrain a linear combination of gridline values to be less than a constant d.

- **Fixed Distance Constraints**

$$\sum_{i=1}^{n} c_i \cdot g_i.v = d \quad \neg g_i.f, g_i \in \mathcal{G}_x \cup \mathcal{G}_y, c_i, d \in \mathbb{R}$$

set a linear combination of gridlines values equal to some constant d. A fixed distance constraint is a conjunction of a minimum distance and a maximum distance constraint.

- **Equal Distance Constraints**

$$C_{Equal} := \left\{ \sum_{i=1}^{n} c_i \cdot g_i.v = \cdots = \sum_{j=1}^{m} c_j \cdot g_j.v \,|\, g_i, g_j \in \mathcal{G}_x \cup \mathcal{G}_y, c_i, c_j \in \mathbb{R} \right\}$$

constrain n linear combinations of gridlines values to be equal.

- **Merge Constraints**

$$C_{Merge} := \{(v_0, \mathit{fixed}_0, G_0), \ldots, (v_n, \mathit{fixed}_n, G_n) \,| \\ \forall i, j \in [0, n] \,:\, G_i \subseteq \mathcal{G}_{x/y}, G_i \cap G_j = \emptyset, v_i \in \mathbb{R}, \mathit{fixed}_i \in \{\mathit{true}, \mathit{false}\}\}$$

is a set of triples (v, fixed, G) where each triple expresses the constraint that all gridlines $g \in G$ should be merged in a single destination gridline g with $g.v = v \wedge g.f = \mathit{fixed}$.

Definition 12. The **Layout Problem** is a tuple $L = (P, \mathcal{C}, o)$ where P is a page according to Def. 11 and $\mathcal{C} = \{C_{Min}, C_{Max}, C_{Equal}, C_{Merge}\}$ a set of constraints. The solution space is defined as $\Delta(L) = \mathbb{R}^{|\mathcal{G}_x \cup \mathcal{G}_y|}$. $o : \Delta(L) \times \{\mathcal{C}\} \rightarrow \mathbb{R}^n$ is an objective function that given a solution and the constraint set $\mathcal{C}$ calculates a value corresponding to the quality of the achieved layout.

The next chapter describes a set of elementary operations for the construction of Layout Problem instances. The objective function o is defined in chapter 5 together with an algorithm that, given a Layout Problem instance $L = (P, \mathcal{C}, o)$, finds a solution $d \in \Delta(L)$ such that

$$d = \mathbf{max}_{d' \in \Delta(L)}\, o(d', \mathcal{C}).$$

Chapter 4

Interaction

The preceding chapters have summarized the principles of direct manipulation interfaces and how these principles have been applied to constraint-drawing applications. Successful interfaces reduce the cognitive effort required to achieve a task. They let their users interact directly with objects that have an analogue in the users' mental model of the task at hand.

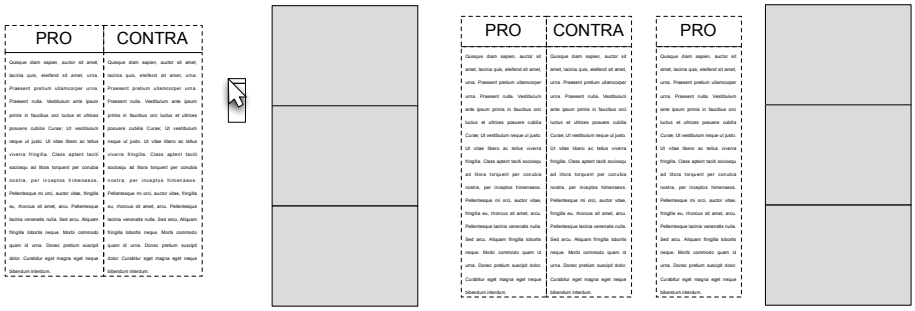
4.1 The ICBM User Interface

How can the user interface requirements laid out at the end of chapter 2 be met? An interactive layout application needs persistent constraints to represent the layout features that the user wants to maintain. As stated before, the available constraints must represent simple concepts and their meaning must suggest a simple visualization. Hence, in the ICBM system the available set of explicit constraints is limited. Users can fix a shape's absolute position, its width and height, and several shapes can be constrained to have the same width or height.

4.1.1 Defining a Partial Gridline Order

Figure 4.1a shows a table with two columns on the left and a stack of objects on the right that could be, e.g., charts or images. The user has selected the left column and has copied it to the clipboard. After the user has pressed Ctrl-V to paste the shape from the clipboard, the small insertion indication shown between the table and the object stack is drawn and it follows the mouse cursor. It depicts a miniature version of the copied column.

If the user presses the mouse button again, thus inserting the shape at the current mouse position, the result would look like Fig. 4.1. The new



(a) (b)

Figure 4.1: Inserting a table column.

column has been inserted between the table on the left and the object stack on the right. The user has only specified the spatial relation that the new column be placed between the table and the stack. The vertical position of the column relative to the object stack has not been specified at all. The layout algorithm has calculated the absolute position and size of the shape.

Previous approaches confused the responsibilities of the user and the layout system because both could move and resize objects. In the ICBM system, the user input is only used to deduce the desired spatial relations between the shapes. This makes layouts *reproducible*. Of course, the user can control a shape's size and position. In that case the user's decision will be respected by the layout algorithm and it remains always unchanged. Thus, the user's expectation is always clear: His input is either left completely unchanged, or size and position will certainly change.

4.1.2 Alignment

Instead of inserting the new column between the table and the object stack, the user will probably want to align the new column with the table. In Fig. 4.2 the insertion indication has snapped to the top and right gridlines of the existing table. When the mouse cursor is within a fixed distance t around a gridline g the insertion indication snaps to g, i.e., the insertion indication remains stuck at gridline g. The snapped gridlines are highlighted to indicate that the shapes will be aligned. Snap-dragging facilitates the establishment of alignment constraints, the most important spatial constraints that users can specify. In the example, the top left of the new column is aligned to the top right of the existing table. This alignment would be most desirable if the user wants to extend the table by a third column.

If the user presses the mouse button, keeps it pressed and drags the mouse, the user can specify the two opposite points of the insertion rectangle as in Fig. 4.2b. This way, the insertion indication can snap both to the top and to the bottom of the existing table. Releasing the mouse button yields the result shown in Fig. 4.2c.

Again, the user did not specify the width, height or absolute position of the new table column. The layout algorithm is responsible for finding a layout in which all columns have equal widths if possible. They are required to have the same height because the table columns are aligned to each other, the exact height is however determined automatically. The space between the table and the stack is also calculated by the layout algorithm depending on the available page space and the amount of content in the table.

The insertion indication can also be snapped to the single gridline between the two columns as in Fig. 4.3a. The user had first aligned the insertion indication with the top of the center gridline and then dragged the mouse cursor down until it snapped to the bottom of the center gridline. If the table column was inserted at this position, its left and right coordinate would be identical and the column would have zero width. Instead, the new column is *squeezed* into the infinitely small space between the two existing columns. The result is shown in Fig. 4.3b. The columns are moved apart and the inserted column is enlarged by the layout algorithm.

4.1.3 Removing Alignment Constraints

In the ICBM system, the alignment of several shapes is represented by attaching all shapes to the same gridline. If the shapes are aligned and one of them is dragged by the user, this shape is automatically removed from the alignment constraint. ICBM constraints are internal representations of user intentions. They force the layout algorithm to do what the user wanted. The constraints do not restrict the user. The system should behave as if it was a common diagram drawing software: If the user has selected a single shape only this shape will be manipulated, i.e., all manipulations should only have *local* effects. In general, the user may select and manipulate multiple shapes. All constraints over the manipulated shapes will be maintained, the constraints over the untouched shapes will also remain unchanged, but constraints between both sets of shapes can be deleted if otherwise they become unsatisfied.

Thus, if a user interaction violates a previously established constraint, the ICBM system will silently remove this constraint. The ICBM system can also add constraints that are necessary to maintain important layout properties that the user expects not to change. Layout constraints in the ICBM system

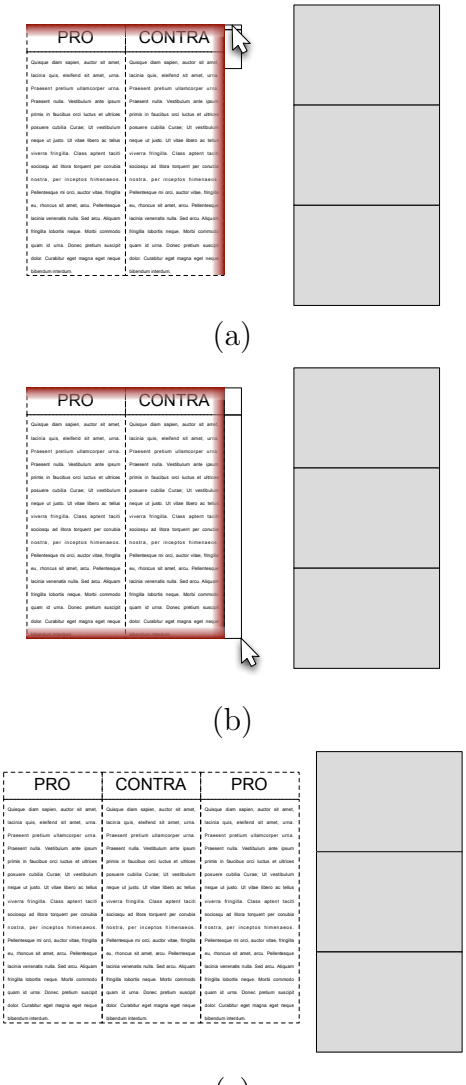
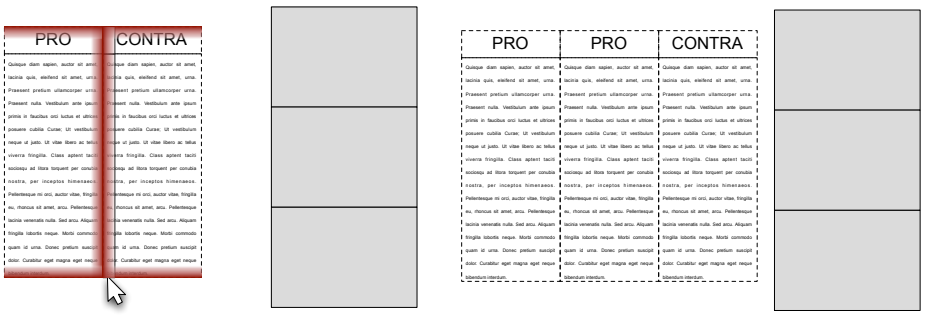


Figure 4.2: Inserting a table column with multiple rows.



(a) (b)

Figure 4.3: Squeezing a new column into a table.

have weaker semantics than constraints in the CAD applications mentioned before. Constraints represent important layout properties but they are only internal to the layout system and can therefore be changed at any time.

4.1.4 Over-constrained User Input

ICBM constraints are weaker in another aspect. The constraint solution algorithm leaves constraints unsatisfied if the layout problem is otherwise infeasible. All constraints are so-called soft constraints as the next chapter will explain in greater detail. Their degree of satisfaction is maximized in the order of constraint importance. Thus, the ICBM layout system can not only solve over-constrained systems of constraints but the solution visualizes which parts of the layout are too constrained.

4.1.5 Under-constrained Inputs

Inputs are usually under-constrained. In the first example above, the vertical position of the newly inserted column is completely unconstrained. To disambiguate this situation, the layout system could create additional constraints. Previous constraint applications used the insertion position as a means to disambiguate the solution. Alternatively, the ICBM system could enforce not just a partial but a total order on all gridlines. If, in Figure 4.1, the new column would be inserted at the same position, the column would be limited to the space between the top of the table and the bottom of the first object on the right. Early prototypes of the ICBM system implemented this strategy but as the example shows, the resulting interface felt overly rigid and it enforced unintended constraints. In the above example, the ta-

ble columns and the object stack should be able to move freely in vertical direction without affecting each other, although it is probably preferable that both are centered vertically. If text was added to the table, its height should increase until it eventually surpassed the height of the stack. If a total order was imposed on the gridlines in vertical direction, the table could not grow in vertical direction unless the object stack's height increased too. No user expects this outcome.

The ICBM system employs an alternative technique that detects under-constrained solutions only if they negatively affected the solution a posteriori. This is based on two observations: First, it is hard to detect true underspecification. The spatial relation between two shapes that are not part of a single constraint may in fact be uniquely determined by a transitive chain of constraints. Second, it is not obvious how an under-constrained solution should be resolved a priori. The solution developed in this thesis is presented in detail in section 5.3 of chapter 5.

4.1.6 Designing the Snap Interaction

Persistent alignment constraints are the most important constraints in the ICBM system and are very easy to establish. When the user drags the corner of an object towards another object, once the user is within a fixed distance of the target object, the dragged corner snaps to the target object's outline.

From the user's perspective, there are several possibilities how snapping to gridlines could work that are illustrated in Fig. 4.4. The simplest idea is that the mouse cursor snaps to a single gridline when the distance between cursor and gridline is less than some constant tolerance. As illustrated in Fig. 4.4a the obvious drawback is the lack of precision. In any layout, different gridlines may coincide without being explicitly aligned. The layout solver placed gridlines g_x^0, g_x^1, g_x^2 at the same coordinate. With a solver that favors an overall regular layout such a result can be expected frequently. Thus the user cannot select precisely which gridline should be snapped to. Because the gridlines coincide but are not identical, the solver may also move them apart again, yielding the unexpected result of Fig. 4.4d.

A small change to the snapping algorithm solves the problem of unpredictability, as illustrated in 4.4b. Instead of taking a random gridline from the set of coinciding gridlines, this approach snaps to the gridline that maximizes the overlap between its occupied spans and the span of the dragged gridline. While this approach is more predictable, the result of Fig. 4.4e is just as bad as the previous one.

Coinciding gridlines are indistinguishable from explicitly aligned gridlines. The solution, shown as Fig. 4.4c, is to snap to all coinciding gridlines at once.

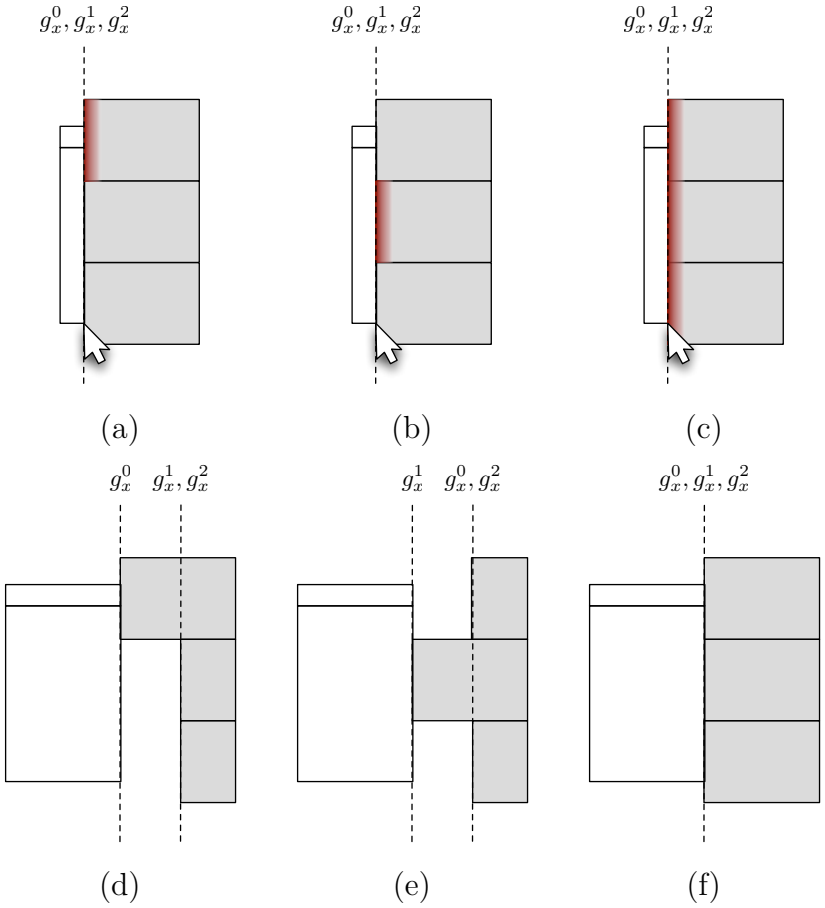


Figure 4.4: Different snapping strategies that were explored: (a) single gridline snapping (b) single gridline snapping with target selection (c) snapping to multiple gridlines with the result shown below in (d-f).

The user has never required the coinciding gridlines to be aligned but they look as if they were. This mistaken perception creates a strong expectation that the ICBM system tries to fulfill. The snapping algorithm inserts the alignment constraints that the user has silently assumed from looking at the layout. This illustrates the weak semantics of constraints in the ICBM system. The system tries to deduce as many constraints as possible because the users cannot and will not think in terms of constraints.

Usually, snapping is a modal operation, i.e., the cursor always snaps if it is within the snapping tolerance unless the user presses a modifier key, in PowerPoint the Alt-key, to override the snapping. Functionality that is only accessible by using modifier keys is difficult to discover. As long as two

gridlines are perceived as visually separate, a space between the two gridlines should be reserved, in which the cursor snaps to neither gridline. In other words, for each pair of adjacent gridlines, the snapping tolerance is adjusted, so that it is always at most a third of the distance between the two gridlines. This is somewhat similar to the snap-and-go technique [10] that overcomes the modality of snapping by moving screen objects slower the closer they get to the snap target.

4.2 Shape Insertion

The basic operation that a user needs to perform is to insert new shapes either by choosing a shape type from the shape menu or by duplicating a shape like in Fig. 4.1. All algorithms in this chapter need an in-memory representation of shapes and their associated gridlines, i.e., of shapes and gridlines that are currently not contained in any page.

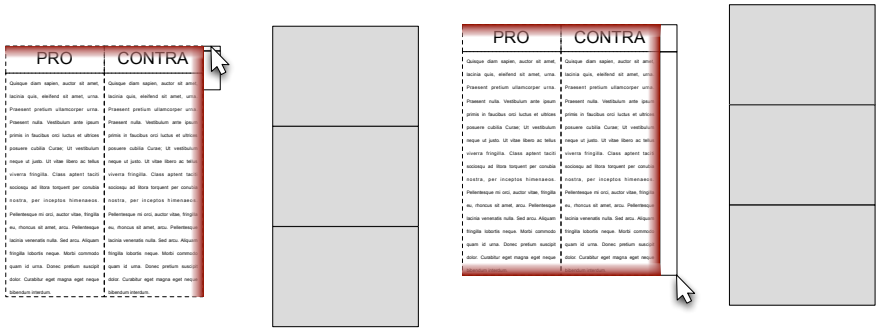
Definition 13. A **source gridline** is defined by a tuple $\hat{g} = (v, o)$ where $v \in \mathbb{R}$ is the coordinate or *value* of the source gridline in either horizontal or vertical direction and $o \in \mathcal{G}_{x/y} \cup \{\circ\}$ is either the original gridline or $\circ$ if this source gridline has no original gridline.

A source gridline list is defined as a gridline list of Def. 9 The source gridline list in x- or y-direction is denoted $\hat{\mathcal{G}}_x$ and $\hat{\mathcal{G}}_y$ respectively, with $\hat{\mathcal{G}} := \hat{\mathcal{G}}_x \cup \hat{\mathcal{G}}_y$.

Definition 14. A **source shape** is a tuple $\hat{s} = (\hat{G}_x, \hat{G}_y, \mathit{span}_x, \mathit{span}_y)$ where $\hat{G}_x$ is a horizontal source gridline list and $\hat{G}_y$ is a vertical source gridline list. The functions $\mathit{span}_x, \mathit{span}_y$ are defined as in Def. 10 but over source gridline lists $\hat{G}_x$ and $\hat{G}_y$.

Definition 15. A **clipboard** is defined as a set of source shapes $C = \hat{\mathcal{S}}$. The set of source gridlines $\hat{\mathcal{G}}_x(C)$, or $\hat{\mathcal{G}}_x$ for short, is defined as $\hat{\mathcal{G}}_x(C) = \{\hat{s}.\hat{G}_x \mid \hat{s} \in \hat{\mathcal{S}}\}$. $\hat{\mathcal{G}}_y$ is defined correspondingly.

A selection of shapes is transformed into a clipboard when it is copied or duplicated. Source shapes and source gridlines are almost identical to their "originals" with the exception that they do not exist on a page but exclusively on the clipboard. A source gridline remembers the original gridline it has been created from as long as that gridline exists. They continue to exist as part of the clipboard after the original shapes have been deleted. Most importantly, the spatial relations between all shapes is preserved when they



(a) (b)

Figure 4.5: (a) Specifying a single snapped location or (b) two opposite points of the insertion rectangle.

are converted to source shapes. When the user inserts the source shapes on a page, the source gridlines are transformed into gridlines again and the source shapes into real shapes. During this transformation, source gridlines may be merged with previously existing gridlines on the page, thereby merging the spatial relations of the source shapes with those of shapes on the page. When a shape is inserted directly from the shape menu, a set of source gridlines and a single source shape can be created automatically depending on the type of shape inserted.

4.2.1 Insertion Modes

The top-level shape insertion algorithm is shown as Alg. 1. It implements a state machine to distinguish the two insertion modes displayed in Fig. 4.5. The details of snapping and inserting the gridlines and shapes are delegated to algorithms that are implemented identically in x- and y-direction. The procedures are either passed the x-coordinate and the corresponding $\mathcal{G}_x$ or the y-coordinates and $\mathcal{G}_y$ as arguments.

Initially, when the user selects a shape from the menu or duplicates a shape and moves the mouse over the page, the algorithm is in state *MouseMove*. The small insertion indication of Fig. 4.5a is shown. By moving the mouse, the user only specifies a single shape corner. The shape will be inserted in an ϵ-sized rectangle at the specified point. The user may align this corner to a single gridline as the illustration shows. If the mouse button is immediately released after it has been pressed, the algorithm's state changes to *ButtonUp* and the shape is inserted. If the mouse is pressed but not released, the algorithm remains in state *ButtonDown*. One corner of the insertion rect-

Algorithm 1: Shape insertion algorithm

```
1  begin
2      s ← MouseMove
3      while s ≠ ButtonUp do
4          p ← current mouse coordinate, p ∈ ℝ²
5          e ← mouse state, e ∈ {MouseMove, ButtonDown, ButtonUp}
6          if e = ButtonDown then
7              ptButtonDown ← p
8              s ← ButtonDown
9          else if e = MouseMove then
10             if s = ButtonDown then
11                 Snap(𝒢_x, ptButtonDown.x, p.x)
12                 Snap(𝒢_y, ptButtonDown.y, p.y)
13             else
14                 Snap(𝒢_x, p.x, p.x)
15                 Snap(𝒢_y, p.y, p.y)
16             end
17         else if e = ButtonUp then
18             InsertGridlines(𝒢_x)
19             InsertGridlines(𝒢_y)
20             MergeGridlines(𝒢_x)
21             MergeGridlines(𝒢_y)
22             SplitGridlines(𝒢_x)
23             SplitGridlines(𝒢_y)
24             s ← ButtonUp
25         end
26     end
27 end
```

angle has been fixed to point *ptButtonDown* and the user may then specify the opposite point by moving the mouse as shown in Fig. 4.5b. Once the mouse is released, the opposite point is fixed as well, potentially snapped to another gridline, and the algorithm's state is changed to *ButtonUp*.

Once the mouse has been released and state *ButtonUp* is reached, the shape and gridlines are inserted in procedure *InsertGridlines*, the snapped gridlines are merged in *MergeGridlines* and *SplitGridlines*, as implied by the name, splits the snapped gridline as Fig. 4.1e showed.

4.2.2 Snapping Algorithm

The snapping algorithm shown in Alg. 2 is passed three arguments: the gridline list $G_{x/y} \subseteq \mathcal{G}_{x/y}$ of snap targets and the interval of extremal values (*cButtonDown*, *cDrag*) that the user has defined.

Definition 16. A **gridline location** is a pair $gl = (v, G)$ where $v \in \mathbb{R}$ is the value and $G \subseteq \mathcal{G}_{x/y}$ is a possibly empty set of snapped gridlines at this value such that $\forall g_i \in G : v = g_i.v$.

Definition 17. A **source gridline mapping** is a set of pairs

$$M = \left\{ (\hat{g}_i, g_j) \,|\, \hat{g}_i \in \hat{\mathcal{G}}_{x/y}, g_j \in \mathcal{G}_{x/y} \right\}$$

assigning to a subset of source gridlines a gridline with which they should be merged.

First, function ClosestGridlines shown in Alg. 3 calculates gridline location $gl_l = (l, G_l)$ at *cButtonDown*, the value where the user has pressed the mouse. *cDrag* is snapped to gridline location gl_r. Gridline locations gl_l and gl_r define the gridline mapping for the extremal source gridlines $\hat{G}_l$ and $\hat{G}_r$. Finally, the interior source gridlines are snapped with algorithm *SnapInteriorGridlines* described in detail in section 4.2.3.

Finding the set of closest gridlines

Given a gridline list G, a tolerance t and a value v, function ClosestGridlines shown in Alg. 3 returns a gridline location $gl = (s, G_s)$ with snapped gridlines $G_s \subseteq G$ within a $2t$ interval around v. If G_s is empty, then $s = v$. The algorithm uses the fact that a gridline list is ordered according to Def. 9. Thus, finding the gridline with smallest index i and value greater than or equal to v can be done in logarithmic time. The algorithm then proceeds by calculating the difference between v and the gridlines g_i and g_{i-1}. Neither is

Algorithm 2: Snap algorithm

Input: Gridlines $G_{x/y} \subseteq \mathcal{G}_{x/y}$, coordinates *cButtonDown* and *cDrag*
Output: Two gridline locations $gl_l = (l, G_l)$, $gl_r = (r, G_r)$, a gridline mapping M and a snap direction $d \in \{low, none, high\}$

begin
  $t \leftarrow$ *snapping tolerance constant in screen coordinates*
  $gl_l \leftarrow \text{ClosestGridlines}(G_{x/y}, cButtonDown, t)$
  $gl_r \leftarrow \text{ClosestGridlines}(G_{x/y}, cDrag, t)$
  $\hat{G}_l \leftarrow \{\hat{g}_l \mid value(\hat{g}_l) = \min_{\hat{g} \in \hat{G}_{x/y}}(\hat{g}.v)\}$
  $\hat{G}_r \leftarrow \{\hat{g}_r \mid value(\hat{g}_r) = \max_{\hat{g} \in \hat{G}_{x/y}}(\hat{g}.v)\}$
  $d \leftarrow none$
  if $l = r$ **then**
    if $cDrag < l$ **then** $d \leftarrow low$ **else** $d \leftarrow high$
  end
  $M \leftarrow \emptyset$
  if $G_l \neq \emptyset$ **then** $M \leftarrow M \cup \bigcup_{\hat{g} \in \hat{G}_l} (\hat{g}, g_l),\ g_l \in G_l$
  if $G_r \neq \emptyset \wedge l \neq r$ **then** $M \leftarrow M \cup \bigcup_{\hat{g} \in \hat{G}_r} (\hat{g}, g_r),\ g_r \in G_r$
  $M \leftarrow M \cup \text{SnapInteriorGridlines}(G, (l, G_l), (r, G_r))$
end

Algorithm 3: ClosestGridlines algorithm for finding the closest gridlines around a value within a snapping tolerance

Input: Gridlines G, snapping tolerance t, value v
Output: Snapped value $s \ : \ s \in [v-t, v+t]$, snapped gridlines $G_s \subseteq G \ : \ \forall g \in G_s : g.v = s$

```
1  begin
2      s ← v
3      G_s ← ∅
4      i ← min({j | g_j ∈ G ∧ g_j.v ≥ v})
5      d_{i-1} ← ∞
6      d_i ← ∞
7      if 0 < i then
8          d ← v − g_{i-1}.v
9          if d ≤ 2t then d_{i-1} ← d
10     end
11     if i < |G| then
12         d ← g_i.v − v
13         if d ≤ 2t then d_i ← d
14     end
15     t' = t
16     if d_{i-1} ≠ ∞ ∧ d_i ≠ ∞ then t' ← min(t', d_{i-1}/3 + d_i/3)
17     if d_{i-1} < d_i then
18         if d_{i-1} ≤ t' then
19             s ← g_{i-1}.v
20             G_s ← {g | g ∈ G ∧ g.v = s}
21         end
22     else if d_i ≠ ∞ then
23         if d_i ≤ t' then
24             s ← g_i.v
25             G_s ← {g | g ∈ G ∧ g.v = s}
26         end
27     end
28 end
```

guaranteed to exist as v can be larger than the maximum value of a gridline in G or smaller than the minimum value of any gridline.

Between two gridlines, there must always be a space were the mouse cursor does not snap to either of the gridlines, as explained in section 4.1.6. If a gridline with value $a \in (v-t, t+t)$ exists, it can only be snapped to if no other gridline with a value b and, without loss of generality, $a < b$ exists, so that $a + \frac{b-a}{3} \leq v < b - \frac{b-a}{3}$. It follows that for this condition to hold, $b \leq v + 2t$ must also hold.

Proof. From the definition of a it follows that $v-t \leq a \leq v$ and thus $v-a \leq t$.

$$a + \frac{b-a}{3} \leq v \tag{4.1}$$
$$\frac{b-a}{3} \leq v - a \tag{4.2}$$
$$b - a \leq 3(v-a) \tag{4.3}$$
$$b \leq 3v - 2a = v + 2(v-a) \tag{4.4}$$
$$b \leq v + 2t \tag{4.5}$$

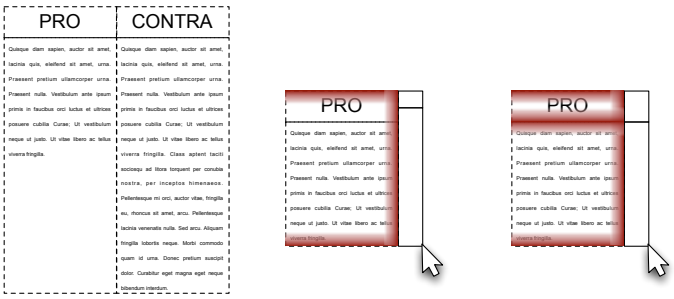
□

Therefore in lines 9 and 13, gridlines g_i and g_{i-1} are accepted, if their distance from v is within a $2t$-interval. In line 16, the tolerance t' is then calculated as the minimum of t or a third of the distance between g_{i-1} and g_i. In other words, if v was in the center third between the two gridlines g_{i-1} and g_i, then ClosestGridlines would return neither g_{i-1} nor g_i.

4.2.3 Snapping Interior Gridlines

Once the user has specified the extremal gridline locations $gl_l = (l, G_l)$ and $gl_r = (r, G_r)$, the destinations of interior source gridlines can be linearly interpolated. In a preliminary version of the layout application, interior source gridlines were snapped like extremal gridlines. However, with many gridlines on a page the high probability that source gridlines snapped unintendedly severely impacted the usability of the feature.

In the example of Fig. 4.6a, the right column is copied to the clipboard and then deleted, afterwards the column is pasted again onto the page as shown in Fig. 4.6b. Because the cell contents of the right column defined the height of the table, the row height changes after it is deleted. If the deleted column is pasted again, the rows do not align anymore as they did before. Thus, it is not possible to recreate the table of Fig. 4.6a simply by aligning the



(a) (b) (c)

Figure 4.6: Snapping interior gridlines to their original gridlines.

columns. In general, the absolute locations of gridlines are a poor indication of their semantics. Instead, the snap algorithm should exploit relationships between gridlines to snap them to each other.

In its simplest form, semantical snapping means that source gridlines snap to their previously aligned gridlines. In the given example, the snap algorithm could discover that the user has snapped two columns to each other that have previously been aligned, i.e., they had been aligned when the right column was copied to the clipboard. This applies to interior gridlines too as Fig. 4.6c shows.

Formally, let $\hat{g}_l$ and $\hat{g}_r$ be the extremal source gridlines in, without loss of generality, x-direction, i.e., $\hat{g}_l := \text{argmin}_{\hat{g} \in \hat{\mathcal{G}}_x}(\hat{g}.v)$ and $\hat{g}_r$ defined correspondingly. If

$$\forall \hat{g} \in \hat{\mathcal{G}}_x : \hat{g}.o \neq \circ \tag{4.6}$$

$$\wedge\ \hat{g}_l.o \in G_l \tag{4.7}$$

$$\wedge\ \hat{g}_r.o \in G_r \tag{4.8}$$

$$\wedge\ \forall \hat{g}_i, \hat{g}_{i+1} \in \hat{\mathcal{G}}_x : \hat{g}_i.v < \hat{g}_{i+1}.v \leftrightarrow \hat{g}_i.o.v < \hat{g}_{i+1}.o.v \tag{4.9}$$

the gridline mapping M can be defined as $M := \{(\hat{g}, \hat{g}.o) \mid \hat{g} \in \hat{G_x}\}$.

In Figure 4.6b, the extremal source gridlines are snapped to the original extremal gridlines. Conditions 4.7 and 4.8 are thus satisfied. In addition, equation 4.6 requires that all original gridlines must still exist – an obvious precondition if the source gridlines shall be snapped to their original counterparts. Equation 4.9 requires that the original gridlines must not have changed their order. This condition ensures that in the resulting layout the original gridlines will have changed their position only, not their order.

4.2.4 Inserting Gridlines

Given the source gridline mapping M, a new gridline has to be created for every source gridline $\hat{g}$, $(\hat{g}, .) \notin M$. The source gridline list $\hat{\mathcal{G}} = (\hat{g}_0, \ldots, \hat{g}_n)$ is sorted as are the existing gridlines $\mathcal{G}$. The mapping M must maintain the source gridline order and the order of existing gridlines must not change. Given two source gridlines $\hat{g}_i, \hat{g}_j, i < j, (\hat{g}_i, g_i) \in M, (\hat{g}_j, g_j) \in M$, all source gridlines $\hat{g}_k, i < k < j$ must be assigned new gridlines g_k so that $g_i < g_k \leftrightarrow \hat{g}_i < \hat{g}_k$ and $g_k < g_j \leftrightarrow \hat{g}_k < \hat{g}_j$. The source gridline list $\hat{\mathcal{G}}$ is reduced to a list of values

$$V = (v_0, \ldots, v_n) \text{ with } v_i = \hat{g}_i.v.$$

The mapping M is transformed into a sorted list of snapped gridlines

$$X = (x_0, \ldots, x_n) \text{ where } x_i = g \text{ if } (\hat{g}_i, g) \in M, x_i = \circ \text{ otherwise.}$$

The list of destination values for each source gridlines is defined as

$$\begin{aligned} D = (d_0, \ldots, d_n) \text{ where } & d_i = x_i.v \text{ if } x_i \neq \circ, \\ & d_i = (v_i - v_k) \cdot \frac{x_j.v - x_k.v}{v_j - v_k} + x_k.v \\ & \text{for } x_j \neq \circ, x_k \neq \circ, k \leq i < j \text{ otherwise.} \end{aligned}$$

Algorithm 4 calculates the interpolated destination positions D. For two subsequent source gridlines $\hat{g}_i, \hat{g}_{i+1} \in \hat{\mathcal{G}}$ with $\hat{g}_i.v < \hat{g}_{i+1}.v$, thus $v_i < v_{i+1}$, the destination positions may have collapsed, i.e., $d_i = d_{i+1}$.

Procedure *InsertGridlines* calculates a gridline list $O = (o_0, \ldots, o_n)$ from sets V, X, D. The gridlines in O maintain the ordering relations of the source gridlines in $\hat{\mathcal{G}}$. When a source shape from the clipboard

$$\hat{s} = (\hat{G}_x, \hat{G}_y, \mathit{span}_x, \mathit{span}_y) \in C$$

is transformed into a shape

$$s = (G_x, G_y, \mathit{span}_x, \mathit{span}_y) \in P$$

on the page, its source gridlines $\hat{G}_{x/y} = (\hat{g}_{i0}, \ldots, \hat{g}_{ik})$ are replaced with gridlines $G_{x/y} = (o_{i0}, \ldots, o_{ik}), o_{ij} \in O$. *InsertGridlines* iterates over all input lists, which all have the same length, in parallel using index i. For each destination value $D[i]$, the algorithm advances index j to the last index so that $D[i] = D[j]$. Two gridlines g_{prev} and g_{next} point to the existing gridlines to the immediate left and right respectively of destination value $D[i]$.

Algorithm 4: Interpolates the destination positions between each pair of snapped gridlines

Input: A sorted list of source values V, a list of snapped gridlines X
Output: A list of interpolated destination values D

begin
  $D \leftarrow \emptyset$
  $i \leftarrow 0$
  while $i < |X|$ **do**
    $j \leftarrow$ find, if it exists, smallest $j > i$ such that $X[j] \neq \circ$, otherwise set $j = |N|$
    if $X[i] \neq \circ \wedge j < |N| \wedge X[j] \neq \circ$ **then**
      define function $d(x) := (x - V[i]) \cdot \frac{V[j]-V[i]}{X[j].v-X[i].v} + X[i].v$
    else
      if $X[i] \neq \circ$ **then**
        define function $d(x) := X[i].v$
      else
        define function $d(x) := X[j].v$
      end
    end
    foreach $k \in [i, j)$ **do** $D \leftarrow D \cup d(V[k])$
    $i \leftarrow j$
  end
end

For every k, $i \leq k \leq j$ where $X[k] \neq \circ$, i.e., where there is a valid snapped gridline, the source value $V[k]$ must be equal. The snap algorithms guarantee that two source gridlines with different source positions cannot snap to the same gridline, or even to different gridlines at the same location as this would change the relation between source gridlines. All such k have the same destination value $D[k]$ by construction, as described above, thus all gridlines $X[k]$ must be at position $D[i] = D[k] = D[j]$. This follows from Alg. 4. *InsertGridlines* must not change the relation among existing gridlines, i.e., after it has run, the gridlines $X[k]$ must still be all at the same position.

It follows that the list of source gridlines $(\hat{g}_i, \ldots, \hat{g}_j)$ can be divided into three lists $((\hat{g}_a, \ldots, \hat{g}_b), (\hat{g}_c, \ldots, \hat{g}_d), (\hat{g}_e, \ldots, \hat{g}_f))$, so that:

$$\forall i \in (a, b), j \in (e, f) : X[i] = \circ \wedge X[j] = \circ$$
$$\forall i \in (a, b), k \in (c, d), j \in (e, f) : V[i] < V[c] = V[k] = V[d] < V[j].$$

Algorithm 5: InsertGridlines algorithm

Input: Gridlines $\mathcal{G}$, source values V, destination values D, snapped gridlines X, $|V| = |D| = |X|$
Output: Gridline list O with $|O| = |X|$

begin
  while $i < |D|$ **do**
    $j \leftarrow$ largest j such that $D[i] = D[j]$
    $g_{\text{prev}} \leftarrow$ last $g_i \in \mathcal{G}$ such that $value(g_{\text{prev}}) < D[i]$
    $g_{\text{next}} \leftarrow$ first $g_i \in \mathcal{G}$ such that $value(g_{\text{prev}}) > D[i]$
    $left \leftarrow 0,\ right \leftarrow 0$
    $snapped \leftarrow X[i]$ is set
    foreach $k = i + 1 \ldots j$ **do**
      if $V[k] \neq V[k-1]$ **then**
        if *snapped* increment *left* **else** increment *right*
      end
      $snapped \leftarrow snapped \vee X[k]$
    end
    $\epsilon_{\text{left}} \leftarrow \min((D[i] - value(g_{\text{prev}}))/(left + 1), 0.01)$
    $gap \leftarrow \min(value(g_{\text{next}}) - D[i], (D[j+1] - D[i])/2)$
    $\epsilon_{\text{right}} \leftarrow \min(gap/(right + 1), 0.01)$
    $rightmax \leftarrow right$
    foreach $k = i \ldots j$ **do**
      if $k > i \wedge V[k] \neq V[k-1]$ **then**
        if $left \geq 0$ **then** decrement *left* **else** decrement *right*
      end
      if *left>0* **then**
        $g \leftarrow$ new gridline at $D[i] - left \cdot \epsilon_{\text{left}}$
      else if *left=0* **then**
        $g \leftarrow X[k]$ or a new gridline at $D[i]$
      else
        $g \leftarrow$ new gridline at $D[i] + (rightmax - right + 1) \cdot \epsilon_{\text{right}}$
      end
      $O[k] \leftarrow g$
    end
  end
end

If any source gridline $\hat{g}_k \in (\hat{g}_i, \ldots, \hat{g}_j)$ is snapped, there is an interval $(c, d) \subseteq (i, j), k \in (c, d)$, so that all source gridlines $\hat{g}_l$ with value $\hat{g}_l.v = \hat{g}_k.v$ have indices in $l \in (c, d)$, and no source gridline with index not in (c, d) is snapped. There may be source gridlines with source values less or larger than $\hat{g}_k.v$.

In lines 8–13, the algorithm iterates over all indices from i up to and including j and counts the different source values in variables *left* and *right*. Both counters correspond to the number of different source values in intervals (a, b) and (e, f). The source gridlines with indices in these two intervals will be moved to destination values left and right of $D[i]$ to reestablish the correct source gridline order in the set of existing gridlines.

The values ϵ_{left} and ϵ_{right} are made small enough so that *left* or *right* gridlines fit to the left or right of $D[i]$ without overlapping the previous or next existing gridline.

The final loop in lines 18–30 iterates over *left* and *right* and creates gridlines at the corresponding indices in gridline list O.

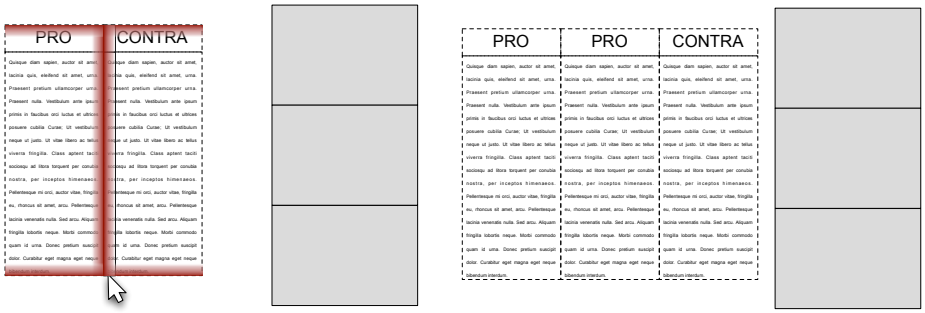
4.2.5 Merging Snapped Gridlines

The extremal snapped gridlines G_l and G_r are merged because this produced the most understandable snapping behavior as detailed in 4.1.6. Since all gridlines in G_l and G_r have the same value, i.e., l or r respectively, merging the gridlines with Alg. 6 is always possible without violating a constraint.

4.2.6 Splitting Gridlines

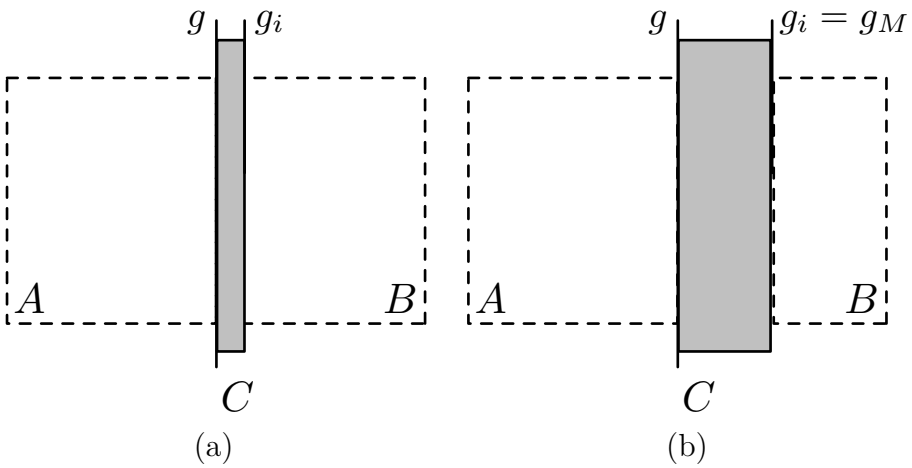
Figure 4.7a shows how the user may insert a table column between the two existing ones. The desired outcome is shown in Fig. 4.7b. If the snapping algorithm Alg. 2 computed extremal gridline locations $gl_l = (l, G_l)$, $gl_r = (r, G_r)$ with $l = r$, depending on the direction d computed by the same algorithm, all source gridlines will have been inserted at positions $l - i\epsilon, i \geq 0$ if $d = \mathit{low}$ or at $l + i\epsilon, i \geq 0$ otherwise. The situation for $d = \mathit{high}$ is shown in Fig. 4.8. The two rectangles A and B are adjacent. They are both bound to gridline g between them. The new rectangle C has been inserted directly on gridline g, i.e., $l = r = g.v$, $g \in G_l$ and $g \in G_r$. Because $d = \mathit{high}$, the gridlines of rectangle C have been inserted right of g. Gridline g_i in the figure is the rightmost gridline of C and its location is $g.v + i\epsilon$.

The algorithm collects the set of gridlines G, bound to the inserted shapes S, the left-most gridline if $d = \mathit{low}$ or the right-most gridline if $d = \mathit{high}$. In the example of Fig. 4.8a, since $d = \mathit{high}$, $G := \{g_i\}$. G thus contains the right-most newly inserted gridlines with position $g.v + i\epsilon$. All gridlines in



(a) (b)

Figure 4.7: Snapping the column on the gridline between two columns inserts the column, thus splitting the gridline.



(a) (b)

Figure 4.8: Separating adjacent shapes to insert a shape on a gridline.

Algorithm 6: MergeGridlines algorithm

Input: A set of gridlines $G \neq \emptyset$ such that $\forall g, g' \in G : g.v = g'.v$
Output: A single merged gridline g_M

```
1  begin
2      g_M ← g ∈ G
3      G ← G\{g}
4      foreach g ∈ G do
5          if g.f then
6              g_M.f ← true
7          end
8          foreach s ∈ S do
9              if g ∈ s.G_{x/y} then
10                 replace g with g_M in s.G_{x/y}
11             end
12         end
13     end
14 end
```

G are merged to a single gridline g_M. Interval set I is set to the interval of extremal values of the span of g_M, i.e., I spans the entire height of shape C. In the example, it follows that $g_M = g_i$.

In the following, every shape $s \in \mathcal{S}\backslash S$ that is attached to a gridline $g_l \in G_l$ is considered. For each shape, I_s is set to the span s covers on g_l in direction d. In the example above, both A and B are attached to gridline g. Because $d = \mathit{high}$, I_s is the span on the right of g. This span is empty for shape A because it touches g from the left. I_s contains a single interval for shape B.

Now, I is the span of g_M, i.e., of the new gridline that is squeezed between A and B. I_s is the span of B at its original position. Because $I \cap I_s \neq \emptyset$, the left side of B should be pushed rightwards to g_M. Intuitively, this means that if C were inserted with zero width at gridline g and then slowly widened, shapes C and B would collide and shape C should push B rightwards as shown in Fig. 4.8b.

Alternatively, if C were smaller, the situation of Fig. 4.9 may arise were $I \cap I_s \neq \emptyset$ and $I \subset I_s$. In this case, shape B is not moved but everything is left as is. As a result, shape C is inserted to the right of g as before but inside B.

Algorithm 7: SplitGridline

Input: Inserted shapes S, gridline location $gl_l(l, G_l)$, insertion direction d

```
1  begin
2      G ← ∅
3      if d = low then
4          G ← {g | g.v = min_{g_i ∈ s.G_{x/y}, s∈S}(g_i.v)}
5      else
6          G ← {g | g.v = max_{g_i ∈ s.G_{x/y}, s∈S}(g_i.v)}
7      end
8      g_M ← MergeGridlines(G)
9      I ← ⋃ span(g_M, d = low ? high : low)
10     if I ≠ ∅ then
11         I ← {(min_{(a',b')∈I}(a'), max_{(a',b')∈I}(b'))}
12         foreach g_l ∈ G_l ∧ ¬g_l.f do
13             foreach s ∈ S\S such that g_l ∈ s.G_{x/y} do
14                 I_s ← s.span_{x/y}(g_l, d)
15                 if I_s ∩ I ≠ ∅ ∧ I ⊄ I_s then
16                     replace g_l with g_M in s.G_{x/y}
17                 end
18             end
19         end
20     end
21 end
```

4.3 Interaction with N Shapes

Although this has not been stated explicitly, all previous algorithms allow inserting, duplicating, and dragging an arbitrary number of shapes at the same time. The table column that served as an example may consist of two text boxes, but it may also be text boxes and a pentagon, text boxes and a column chart or simply four text boxes. All algorithms operate on the source gridline list $\hat{\mathcal{G}}_{x/y}$. The source gridline list directly represents the alignment relations between source shapes.

By snapping the source gridlines to existing gridlines, the alignment constraints between copied shapes attached to the source gridlines are merged with the alignment constraints of existing shapes. This means that a subset $S \subseteq \mathcal{S}$ of all shapes $\mathcal{S}$ can be copied, or removed, and inserted at another location or even on another page. The copied subset S also copies all con-

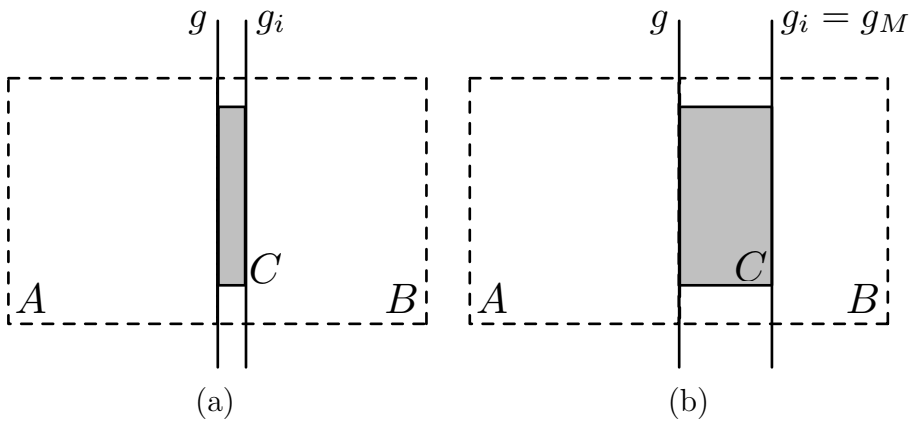


Figure 4.9: The shape is not squeezed between shapes A and B if its span is strictly contained in the span of a neighboring shape.

straints defined between shapes in S. On insertion of S, the constraints defined over shapes S are reintegrated into the page constraint set.

4.4 Manipulating Shapes

After a set of shapes has been inserted successfully, the shapes are selected and can be further manipulated. Selected shapes have a blue outline to be easily distinguished and an arbitrary number of shapes can be selected at the same time. On each individual selected shape, so called *handles* appear on each individual gridline and typically on each point of crossing x- and y-gridlines. Dragging on one of these handles moves the gridline or the gridlines this handle is attached to. The whole set of selected shapes has a single rotation *handle* as shown in Fig. 4.10. The rotation handle is the handle on the very top that resembles a hook.

4.4.1 Dragging Shapes

The algorithms presented so far can be adapted to support dragging existing shapes on the slide. Reusing the previous example, the middle column, pictured below in Fig. 4.11, is selected and dragged around the slide.

The drag algorithm shown in Alg. 8 is a much simplified variant of the insertion algorithm Alg. 1. The action of dragging a shape is initiated only when the mouse is pressed, held and then moved. While the user keeps the mouse button pressed, method *Snap* is called.

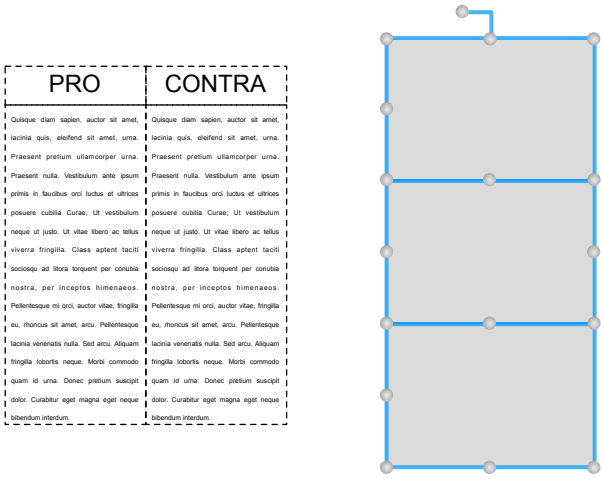
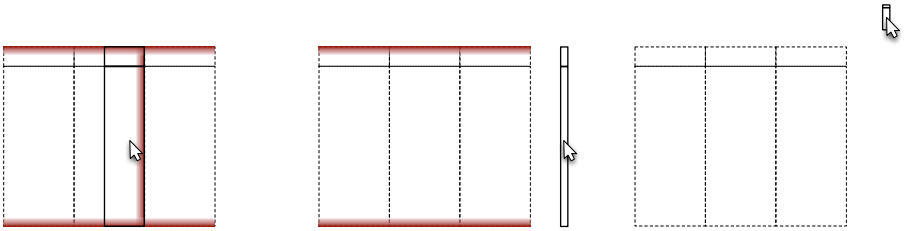


Figure 4.10: Selected shapes have a distinctive outline color and handles allow changing the gridline alignment or position.



(a) (b) (c)

Figure 4.11: The middle column is dragged rightwards (a) until it is right of the table (b) and then it is dragged upwards.

Algorithm 8: Shape dragging algorithm

Input: mouse coordinate $p \in \mathbb{R}^2$ and mouse event $e \in \{\mathsf{MouseMove}, \mathsf{ButtonUp}\}$

```
1 begin
2     if e = MouseMove then
3         Snap(G_x, p)
4         Snap(G_y, p)
5     else if e = ButtonUp then
6         insert shapes identical to Alg. 1
7     end
8 end
```

The snap algorithm in Alg. 9 uses the same basic procedures that are described above to provide a slightly changed behavior. As long as the mouse cursor is moved within the selected shape, e.g., to the right as in Fig. 4.11a, one side of the shape remains at its location while the left side in the example follows the mouse. Once the mouse cursor has left the shape, or if there is no adjacent shape at all, the mouse cursor only specifies a single point as in the default insertion case. Because the cursor is not moved vertically in Fig. 4.11a-b, the column remains vertically aligned with the other two columns. Pressing Shift while dragging a shape only moves the shape either horizontally or vertically. Pressing Ctrl drags a copy of the shape and leaves the original unchanged. If shapes are copied using Ctrl-drag, the copy can stay aligned to the original in either horizontal or vertical direction.

If the user presses shift, the shape is only moved orthogonally in one direction, the other gridlines remaining fixed. Thus, in lines 5-7, the gridlines in one orientation snap to their original positions with zero tolerance. The condition in line 13 tests if any extremal source gridline has an original gridline, i.e., if any other shape is attached to the gridline. If this is the case, either the left extremal coordinate (ll. 14-18) or the right one (ll. 20-23) is kept constant. The left-most coordinate moves with the mouse towards the right if $n_0 \leq n$, i.e., if the mouse is moved right. The snapping tolerance t is scaled linearly with factor *scale*. The further right the mouse is moved, the closer it gets to the right extremal source coordinate and the smaller the snapping tolerance must be. The remainder of the algorithm that calculates the source gridline mapping M is identical to Alg. 2.

Algorithm 9: Snap algorithm used for dragging

Input: Gridlines G, orientation $o \in \{horz, vert\}$, mouse coordinate $p \in \mathbb{R}^2$

Output: Two gridline locations $gl_l = (l, G_l)$, $gl_r = (r, G_r)$, a source gridline mapping M

begin
  $(min, max) \leftarrow (\min_{g' \in \hat{G}}(g'.v), \max_{g' \in \hat{G}}(g'.v))$
  $p_0 \leftarrow$ point where user started dragging
  $v \leftarrow p - p_0$
  if *shift pressed* $\wedge\ ((|v.x| \leq |v.y|) = (o = horz))$ **then**
    $(l, G_l) \leftarrow \text{ClosestGridlines}(G, min, 0)$
    $(r, G_r) \leftarrow \text{ClosestGridlines}(G, max, 0)$
  else
    $\hat{G}' \leftarrow \{\hat{g} \mid \hat{g} \in \hat{G}, \hat{g}.v = min \vee \hat{g}.v = max\}$
    $t \leftarrow$ snapping tolerance constant
    $n \leftarrow o = horz\ ?\ p.x : p.y$
    $n_0 \leftarrow o = horz\ ?\ p_0.x : p_0.y$
    if $\exists \hat{g} \in \hat{G}' : \hat{g}.o \neq \circ \wedge n \in [min, max]$ **then**
      if $n < n_0$ **then**
        $scale \leftarrow (max - min)/(min + n_0)$
        $m \leftarrow min - scale \cdot (min - n)$
        $(l, G_l) \leftarrow \text{ClosestGridlines}(G, min, 0)$
        $(r, G_r) \leftarrow \text{ClosestGridlines}(G, m, t \cdot scale)$
      else
        $scale \leftarrow (max - min)/(max - n_0)$
        $m \leftarrow max - scale \cdot (max - n)$
        $(r, G_r) \leftarrow \text{ClosestGridlines}(G, max, 0)$
        $(l, G_l) \leftarrow \text{ClosestGridlines}(G, m, t \cdot scale)$
      end
    else
      $(l, G_l) \leftarrow (r, G_r) \leftarrow \text{ClosestGridlines}(G, n, t)$
      $d \leftarrow n < l\ ?\ low : high$
    end
  end
  calculate source gridline mapping M as in Alg. 2
end

4.4.2 Shape Rotation

Via the rotation drag handle shown in Fig. 4.10 a single shape or a set of shapes can be rotated and flipped. Rotation and flipping is implemented using the snap algorithm of Alg. 2. If a set of shapes is rotated, the source gridlines are mapped to gridlines in the orthogonal direction, i.e., horizontal source gridlines are mapped to vertical gridlines and vice versa. Similarly, flipping is implemented by flipping the order of source gridlines.

4.4.3 Shape Alignment and Position

Each handle h is attached to at most one gridline in each direction, i.e., corner handles are attached to one x- and y-gridline. By dragging a handle h and thus the attached gridline the user can express different things. Figure 4.12a shows how the left gridline g_l of the object stack is dragged to the left so that it snaps to the table. Assume the handle is moved to coordinate x and let the rightmost gridline of the table be g_2. The user has expressed two different things at the same time: Gridline g_l is moved to location x and g_l is now identical to g_2.

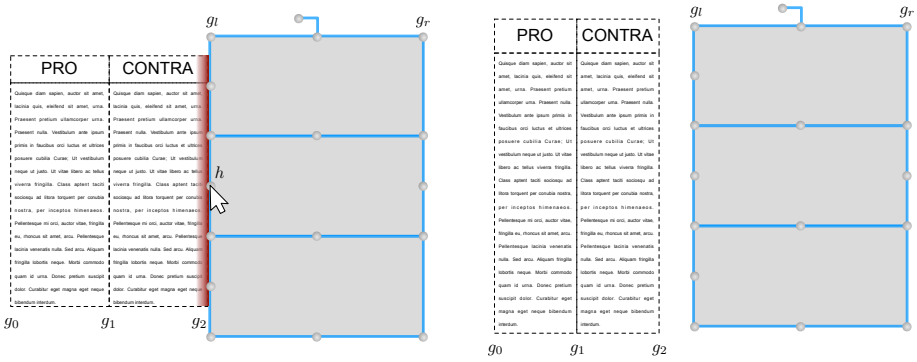


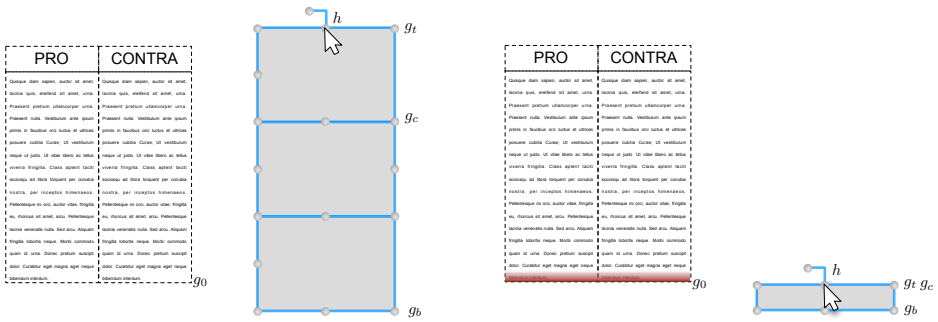
Figure 4.12: (a) Dragging a gridline changes gridline relations or (b) dragging a gridline changes gridline locations.

Changing the relation between gridlines, i.e., unifying g_l and g_2 into a single gridline, changes the way gap constraints are collected. Before, there was a gap between g_2 and g_l that kept the table and the object stack separated. Now, there can be no gap between g_2 and g_l anymore, meaning that the table will directly touch the object stack.

Figure 4.12b shows the alternative result if gridline g_l is fixed at location x *without unifying* g_l and g_2. The object stack has been extended to the left because g_l was moved. The relations between the gridlines have remained unchanged, i.e., there is still a gap between gridlines g_2 and g_l, and therefore the table has moved left. In a constraint-based layout system, the user should rarely need to explicitly fix a gridline's position because the gridline positions should be calculated by the layout system.

When a handle is moved, we distinguish two different modes of specifying gridline *relations* and gridline *locations* as detailed in the interaction principles on page 47. If $G_S = \bigcup_{s \in S} s.G_{x/y}$ are the gridlines bound to the shape selection and G are the gridlines attached to unselected shapes, the default drag mode maintains the relations of gridlines in each set G_S and G but the relations between the two sets may be changed. If the Alt-modifier key is pressed while a handle is moved, the user changes the interaction mode. In this mode, only the position of gridline g_l is fixed. All gridline relations remain constant.

Aligning Gridlines



(a) (b)

Figure 4.13: Selected gridlines move along when the dragged gridline passes them.

The selected shapes $s \in S \subseteq \mathcal{S}$ are attached to a subset of all gridlines in each dimension $G_S = \bigcup_{s \in S} s.G_{x/y} \subseteq \mathcal{G}_{x/y}$. On each gridline $g_i \in s.G_{x/y}$ of each shape $s \in S$ will be at least one handle h, as in the examples of Fig. 4.13. The example shows how the top-most handle h is dragged downwards and snapped to gridline g_0 at the bottom of the table. The handle snaps to gridline g_0 and its occupied spans are highlighted to visualize the snapping.

If the handle is dropped in this situation, gridlines g_0 and g_t will be merged and thus top of the object stack will be aligned to the bottom of the table.

While the handle h is dragged downwards, it passes gridline g_c. According to the principle explained above, all gridline relations inside the set of selected gridlines G_S remain constant. Thus, when g_t moves downwards, it collects all gridlines it passes and gridline g_c is moved downwards too, only separated by an infinitesimal ϵ. Gridline g_b is of course still at its previous position because its relation to any other gridline in G_S has not yet changed. This allows the user to express with a single drag operation that, everything else being equal, two gridlines should be aligned to each other.

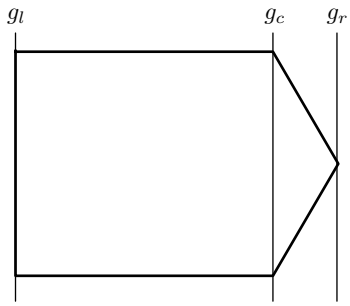


Figure 4.14: Internal gridline g_c moves with the dragged gridline g_r.

When a gridline is dragged, all other selected gridlines are collected and moved too once they are passed by the dragged gridline. For shapes that have internal gridlines like the pentagon does, this functionality alone is not enough. A pentagon has three gridlines in horizontal direction as Fig. 4.14 shows. When the gridline g_r at the pentagon tip is dragged to the right, gridline g_c should follow so that the length of the pentagon tip remains constant.

Therefore Alg. 10 begins by collecting the set F of internal gridlines together with their move factors. In the pentagon example, if dragged gridline g is g_r, F would contain the pair $(g_c, 1)$, meaning that gridline g_c moves in parallel with gridline g_r.

In a selection of multiple shapes, only the gridlines of shape s owning the drag handle are considered, to avoid conflicts between different shapes that would like to move gridlines by a different degree. Then, while the user keeps dragging the handle, the gridline location (v, G_v) is computed in the loop of lines 7-9. As in the previous algorithms, $v \in \mathbb{R}$ is the snapped location and G_v the potentially empty set of gridlines at value v that the user snapped to. When the user has finished dragging, the input to procedure *InsertGridlines* (Alg. 5) is computed, the set of gridlines G' and of snapped gridlines X.

Algorithm 10: Dragging a handle to change gridline alignment

Input: The dragged handle h with the attached dragged gridline g and shape s and all selected shapes S.
Output: Moved gridlines G' with destination positions D and snapped gridlines X

begin
  $F \leftarrow \emptyset$
  foreach $g' \in s.G_{x/y}$ **do**
    // collect all gridlines of s that move along with g by factor f
    $F \leftarrow F \cup (g', f \in \mathbb{R})$
  end
  $(v, G_v) \leftarrow (g.v, \emptyset)$
  while h *is dragged* **do**
    $n \leftarrow$ current mouse position in x-direction
    $(v, G_v) \leftarrow$ ClosestGridlines($\mathcal{G}_{x/y}, n,$ snap tolerance constant)
  end
  $G' := \{g'_0, \ldots, g'_n\} \leftarrow \bigcup_{s' \in S} s.G_{x/y}$
  $V := \{v_0, \ldots, v_n\} \leftarrow \{g'_0.v, \ldots, g'_n.v\}$
  $D \leftarrow V$
  $X := \{x_0, \ldots, x_n\} \leftarrow \{x_i \mid$ if $g'_i.v = g.v \wedge G_v \neq \emptyset$ then $x_i \leftarrow g_v \in G_v,$ else $x_i \leftarrow \circ\}$
  foreach $g'_i \in G'$ *with* $(g'_i, f) \in F$ **do**
    $d_i \leftarrow g'_i.v + f \cdot (g.v - v)$
  end
  foreach $g'_i \in G'$ *with* $(g'_i, .) \notin F$ **do**
    find largest k and smallest l, $k < i < l$, with $(g'_k, .), (g'_l, .) \in F$
    if $g'_k.v \leq g'_i.v \wedge d_k > g'_i.v$ **then**
      $d_i \leftarrow d_k$
    else if $g'_i.v \leq g'_l.v \wedge g'_i.v > d_l$ **then**
      $d_i \leftarrow d_l$
    end
  end
  InsertGridlines (V, X, D)
  MergeGridlines(G_v)
end

If G_v is not empty, all gridlines with the same value as dragged gridline g are snapped to the gridlines G_v. The calculation of the set of destination values D is a bit more complicated. Calculating the destination values for the gridlines in F is straight-forward. For all other gridlines $g'_i \in G'$, the (at most) two gridlines g'_k, g'_l represented in F and surrounding g'_i are searched. It holds that $g'_k.v \leq g'_i.v \leq g'_l.v$ and d_i is set so that $d_k \leq d_i \leq d_l$ holds. In other words, all gridlines that are moved directly by dragging the handle are in F. Their position is calculated first and afterwards all other gridline positions are computed such that the order of all gridlines in G' remains constant.

Moving Gridlines

If the user drags handle h and simultaneously keeps the Alt-key pressed, the position of gridline g is fixed to a specific screen coordinate. Alt is used frequently as a modifier key. In PowerPoint, e.g., it can be used while dragging to disable the snapping. Here, it is used in a similar manner to disable the snapping to gridlines. The visual feedback is exactly the same as in Fig. 4.13. Now, assume handle h is dragged to some value v. Simply setting $g.v \leftarrow v$ may change the relation of gridline g to all other gridlines in $\mathcal{G}_{x/y}$. Setting the gridline location must maintain all gridline relations however. Adding the constraint $g.v = v$ to the constraint set achieves this elegantly. The layout solver will maintain all relations between gridlines that have been defined previously. The layout solver may not be able to fulfill the constraint however, in which case it will be dropped and the drag operation will have had no effect.

Formally, when handle h is dropped at value v, a new gridline g' is created with $g'.v = v$. Then procedure *TryMergeInto* is called, shown in Alg. 11, that manages the partitioning of gridlines to be merged C_{Merge} that will be passed to the layout algorithm.

4.4.4 Size Constraints

By dragging on a handle and keeping the Ctrl-key pressed, the user can insert a size constraint. The Ctrl-key is used in PowerPoint to resize shapes symmetrically around their center. This behavior is mimicked when setting a size constraint. When the user drags handle h attached to gridline g, the gridline $g' \in G_S$, the set of gridlines spanned by the selected shapes S, is searched such that $|g.v - g'.v|$ is maximal. Figure 4.15 shows how the shapes are resized symmetrically around the center between g and g' while the user drags on the handle. Algorithm 12 describes the addition of a size constraint

Algorithm 11: TryMergeInto

Input: A gridline g_i and a destination gridline g_j, a boolean *directed*
Output: Gridline partitioning C_{Merge} of gridlines that must be merged

```
1  begin
2      (v_i, fixed_i, G_i) ← find or create partition in C_Merge with g_i ∈ G_i
3      (v_j, fixed_j, G_j) ← find or create partition in C_Merge with g_j ∈ G_j
4      if i ≠ j then
5          if directed ∧ fixed_i ∧ ¬fixed_j then
6              foreach g ∈ G_i do  g.f ← false
7              fixed_i ← false
8          end
9          if ¬(fixed_i ∧ fixed_j) ∨ v_i = v_j then
10             if ¬fixed_i ∧ fixed_j then
11                 fixed_i ← fixed_j
12                 v_i ← v_j
13             end
14             replace partitions i, j with single entry (v_i, fixed_i, G_i ∪ G_j)
15         end
16     end
17 end
```

$g_r.v - g_l.v = d, g_r, g_l \in \mathcal{G}_{x/y}, d \in \mathbb{R}$. The algorithm manages a partition $P_{\text{Distance}} := \left\{(I_G^i, d_i) \mid I_G^i \subset \mathcal{G}_{x/y} \times \mathcal{G}_{x/y} \wedge \forall i, j, i \neq j : I_G^i \cap I_G^j = \emptyset\right\}$ in which each element is composed of a set of gridline intervals $I_G^i := \{(g_i, g_j), \dots\}$ and a distance value d_i. Every two gridline interval sets I_G^i, I_G^j are mutually exclusive, i.e., no gridline interval can be part of two different size constraints.

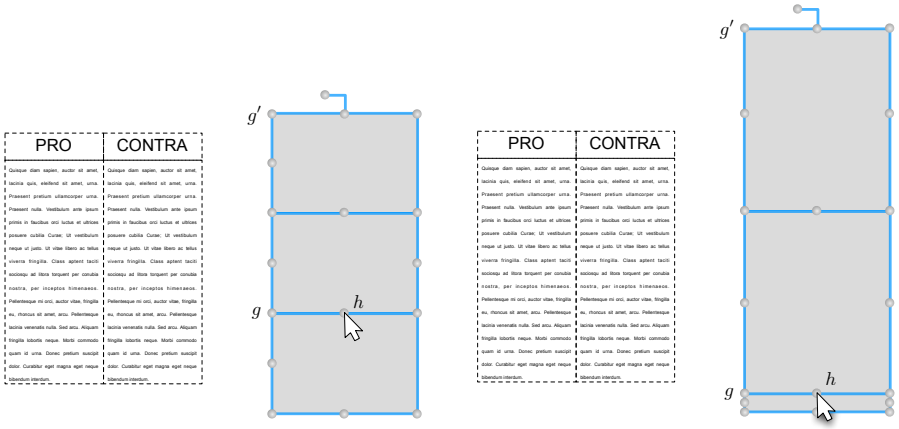


Figure 4.15: Dragging a handle and pressing the Ctrl-key resizes shapes symmetrically.

4.4.5 Same Extent Constraints

The second kind of size constraint the user can specify directly is the *Same Width* or *Same Height* constraint. A set I of n gridline pairs, defined as $I := \{(g_l^i, g_r^i) \mid 0 \leq i < n\}$, can be constrained so that

$$g_r^0.v - g_l^0.v = \dots = g_r^{n-1}.v - g_l^{n-1}.v.$$

The gridline pairs I can be added to P_{Distance} as a single partition and the distance d is set to 'e', indicating that this partition is an n-ary equal distance constraint. What should happen if, e.g., (g_l^0, g_r^0) are already constrained to distance $d \in \mathbb{R}$, i.e., there is a $(I_G, d) \in P_{\text{Distance}}$ with $(g_l^0, g_r^0) \in I_G$? Similarly, there may be a subset of I that already shares an equal distance constraint with other gridline pairs not in I. The conflict is resolved by using partition set P_{Distance}. Algorithm 13 iterates over each gridline pair in $(g_l^i, g_r^i) \in I$ and searches the partition in P_{Distance} that already contains (g_l^i, g_r^i). All overlapping partitions thus found are merged into a single partition, their distance

Algorithm 12: InsertSizeConstraint

Input: A set of selected shapes $S \subseteq \mathcal{S}$, an orientation $orient \in \{horz, vert\}$ and a distance d

Output: A partition of gridlines P_{Distance}

begin

$g : S \times \{horz, vert\} \to G$ defined as

$$g(s, orient) := \begin{cases} s.G_x & \text{if } orient = horz \\ s.G_y & \text{otherwise} \end{cases}$$

$G_l \leftarrow \bigcup_{s \in S} \{g \,|\, g \in g(s, orient) \wedge g.v = min_{s' \in S \wedge g' \in g(s', orient)}(g'.v)\}$

$G_r \leftarrow \bigcup_{s \in S} \{g \,|\, g \in g(s, orient) \wedge g.v = max_{s' \in S \wedge g' \in g(s', orient)}(g'.v)\}$

$g_l \leftarrow \text{MergeGridlines}(G_l)$

$g_r \leftarrow \text{MergeGridlines}(G_r)$

Remove (g_l, g_r) from each I^i_G where $(I^i_G, .) \in P_{\text{Distance}}$

$P_{\text{Distance}} \leftarrow P_{\text{Distance}} \cup \{((g_l, g_r), d)\}$

end

value is maximized, and the value 'e' is defined as less than any $d \in \mathbb{R}$. Partition P_{Distance} is transformed into constraint sets $C_{\text{Min}}, C_{\text{Max}}$ and C_{Equal}:

$$\begin{aligned} C_{\text{Min}} = \{&g_j - g_i \geq d \,| \\ &\exists (I_G, d) \in P_{\text{Distance}} : (g_i, g_j) \in I_G \wedge d \in \mathbb{R}\} \\ C_{\text{Max}} = \{&g_j - g_i \leq d \,| \\ &\exists (I_G, d) \in P_{\text{Distance}} : (g_i, g_j) \in I_G \wedge d \in \mathbb{R}\} \\ C_{\text{Equal}} = \{&g_{j0} - g_{i0} = \cdots = g_{jn} - g_{in} \,| \\ &\exists (\{(g_{i0}, g_{j0}), \ldots, (g_{in}, g_{jn})\}, \text{`}e\text{`}) \in P_{\text{Distance}}\} \end{aligned}$$

4.4.6 Size Constraint Visualization

Size constraints and same extent constraints have to be explicitly visualized otherwise the user would easily forget she had specified them and the layout system behavior may appear unpredictable. The visualization must be easily distinguishable, yet unobtrusive. To limit the visual clutter, only a subset of all gridline pair partitions P_{Distance} is visualized at a time. Given the set of selected shapes S, with the attached gridlines G_S, subset

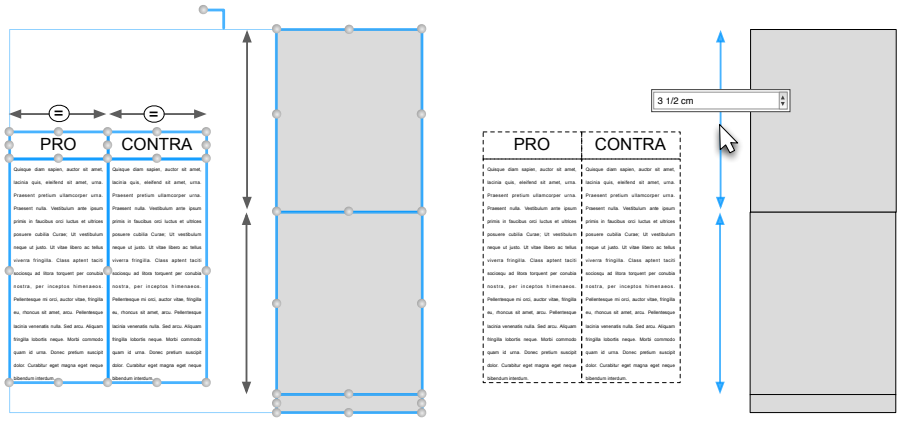
$$\begin{aligned} P := \Big\{&(I^i_G, d_i) \mid (I^i_G, d_i) \in P_{\text{Distance}} \\ &\wedge \exists (g_l, g_r) \in I^i_G : g_l \in G_S \vee g_r \in G_S\Big\} \end{aligned}$$

Algorithm 13: AddSameExtentConstraint

Input: A set of selected shapes $S \subseteq \mathcal{S}$
Output: A partition of gridlines P_{Distance}

begin
  if $|S| > 2$ **then**
    $i \leftarrow -1$
    foreach $s \in S$ **do**
      $(g_l, g_r) \leftarrow$ interval of extremal gridlines of s
      if $g_l \neq g_r$ **then**
        $(I^j_G, d_j) \leftarrow$ find partition with $(g_l, g_r) \in I^j_G$, if no such partition exists, create it with $d_j \leftarrow e$
        if $i = -1$ **then**
          $i \leftarrow j$
        else
          $$d_i \leftarrow \begin{cases} e & \text{if } d_i = e \wedge d_j = e \\ d_i & \text{if } d_j = e \\ d_j & \text{if } d_i = e \\ \max(d_i, d_j) & \text{otherwise} \end{cases}$$
          $(I^i_G, d_i) \leftarrow (I^i_G \cup I^j_G, d_i)$
          update joined partition i in P_{Distance}, remove j
        end
      end
    end
  end
end

is visualized on the screen, i.e., the set containing all size constraints defined over at least one gridline a selected shape is bound to. For each gridline pair $(g_l, g_r) \in (I_G^i, d_i) \in P$ a small arrow is shown. Each arrow can be selected individually or multiple arrows belonging to the same (I_G^i, d_i) can be selected together. For selected arrows, the constraint value, i.e. the exact height or width, can be specified in a small text box. The user may even specify the extent using metric or imperial units such as mm, cm, inches or fractions thereof. The user can insert the string "=" to specify a same extent constraint. Figure 4.16 shows the visualization and Alg. 14 describes the arrow placement algorithm in detail.



(a) (b)

Figure 4.16: (a) Visualizing the size constraints of selected shapes. (b) Editing the constraint value.

4.4.7 Shape Deletion

A selected shape or multiple selected shapes can be deleted by pressing the Delete key. When a shape like the middle grey rectangle of Fig. 4.17a is deleted, it would usually leave a gap between the top and bottom object. If the three rectangles formed a table column and the middle row was removed, the remaining two rows should close the gap between them to form a continuous column again. The same argument would hold, if the middle row was moved and not deleted. If this could be implemented, the user could easily rearrange the columns of a table. The column would be squeezed between two previously adjacent columns with the *SplitGridlines* algorithm and the space where the dragged column used to be would close automatically.

Algorithm 14: Place size constraint visualizations

Input: Set $P \subseteq P_{\text{Distance}}$ of gridline pair partitions, selected shapes S

begin
  foreach $(I_G^i, d_i) \in P$ **do**
    $w_i = \min_{(g_l, g_r) \in I_G^i}(g_r.v - g_l.v)$
    $min_i = \min_{(g_l, g_r) \in I_G}(g_l.v)$
  end
  sort P such that $\forall (I_G^i, d_i), (I_G^j, d_j) \in P$:
  $(I_G^i, d_i) < (I_G^j, d_j) \leftrightarrow w_i < w_j \vee (w_i = w_j \wedge min_i < min_j)$
  $R \leftarrow$ set of bounding rectangles occupied by selected shapes S
  foreach $(I_G^i, d_i) \in P$ **do**
    sort $(g_l, g_r) \in I_G^i$ in ascending order by $g_l.v$
    $x_{prev} \leftarrow \infty$
    $y_{prev} \leftarrow \infty$
    foreach $(g_l, g_r) \in I_G^i$ **do**
      $(l, r) \leftarrow (g_l.v, g_r.v)$
      if $l < x_{prev}$ **then**
        $y_{prev} \leftarrow$ maximum y-value over all rectangles in R
      end
      $x_{prev} \leftarrow l$
      $r \leftarrow$ rectangle spanned by points (l, y_{prev}) and
      $(r, y_{prev} +$ arrow height$)$
      place arrow at rectangle r
      $R \leftarrow R \cup r$
    end
  end
end

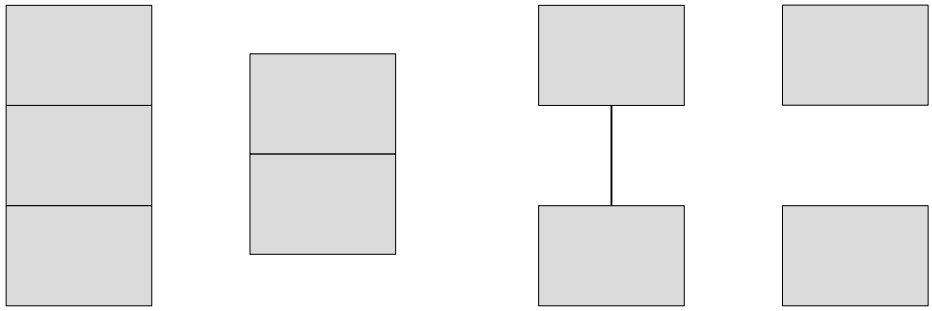


Figure 4.17: (a) When the the center object is deleted, a table should remain compact. (b) Boxes only separated by a connector appear as separate entities and should remain so.

There are cases however, where it might be preferable to keep the gap. In Fig. 4.17b, both text boxes are separated by a connecting line. Visually this leaves the impression that both boxes are really separated and have been connected only for illustrative purposes. The example of Fig. 4.17a shows a table that is supposed to be compact even if a column or row is deleted.

How can these two cases be distinguished? The algorithmic idea is to compare two sets of layout constraints, one gathered before an object is deleted or moved, and one afterwards. If an object disappears between two gridlines and no other object keeps both gridlines apart, then the solver can try to collapse the gridlines by adding a constraint that both gridlines should have an equal value. This merge operation may of course fail because of other constraints.

Algorithm 15 collects the constraint sets $C_{\text{Min}}, C_{\text{Max}}, C_{\text{Equal}}, C_{\text{Merge}}$ before a shape is manipulated and afterwards. By analyzing the difference, the algorithm can decide if two gridlines should be merged using procedure *TryMergeInto* of Alg. 11.

The examples of Fig. 4.18 will serve as running examples throughout the following explanation. In the example of Fig. 4.18a, rectangle B will be deleted. In Fig. 4.18b, rectangles B_0 and B_1 will be removed.

The *CollapseGridlines* algorithm is passed the set of all gridlines $\mathcal{G}_{x/y}$ and begins by collecting for each gridline $g_i \in \mathcal{G}_{x/y}$ the $span^i_{\text{pre}}$. Each occupied interval on this span is annotated by the list of shapes that occupy this interval. In the first example of Fig. 4.18a, the spans are thus:

$$span^0_{\text{pre}} := \{((y_0, y_1), \{A, B\})\}$$
$$span^1_{\text{pre}} := \{((y_0, y_1), \{B, C\})\}\,.$$

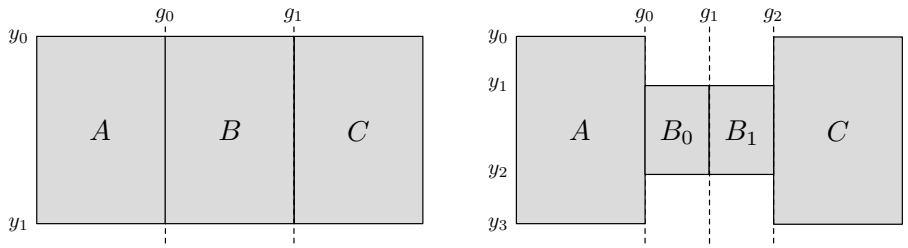


Figure 4.18: Two simple examples to illustrate the CollapseGridlines algorithm.

And for Fig. 4.18b they are:

$$\begin{aligned} span^0_{\text{pre}} &:= \{((y_0, y_1), \{A\}), ((y_1, y_2), \{A, B_0\}), ((y_2, y_3), \{A\})\} \\ span^1_{\text{pre}} &:= \{((y_1, y_2), \{B_0, B_1\})\} \\ span^2_{\text{pre}} &:= \{((y_0, y_1), \{C\}), ((y_1, y_2), \{B_1, C\}), ((y_2, y_3), \{C\})\}\,. \end{aligned}$$

Array $c_{\text{pre}}(i, j) = \mathit{true}$ contains the information that there has been a distance constraint between gridlines g_i and g_j before any action on the shapes has been performed. Distance constraints are transitive so that for all $i < j < k$:

$$c_{\text{pre}}(i, j) \wedge c_{\text{pre}}(j, k) \rightarrow c_{\text{pre}}(i, k).$$

It is important to note that for gridline distance constraints, the transitivity relation only holds for a triple g_i, g_j, g_k with $i < j < k$. For reasons of simplicity, $c_{\text{pre}}(i, j) = c_{\text{pre}}(j, i)$ must also hold.

In the first example, there is only a single constraint guaranteeing a minimum size of shape B and thus $c_{\text{pre}}(0, 1) = \mathit{true}$. In the second example, the size of shapes B_0, B_1 implies $c_{\text{pre}}(0, 1) = c_{\text{pre}}(1, 2) = \mathit{true}$ and by transitivity it follows that $c_{\text{pre}}(0, 2) = \mathit{true}$.

At this point, after $span^i_{\text{pre}}$ and c_{pre} have been set, the selected shapes S, i.e., either shape B or both B_0 and B_1, are dragged or deleted and the second phase of the algorithm begins.

After the shapes have been manipulated, the algorithm collects $span^i_{\text{post}}$ for each gridline g_i. In the first example, the spans of gridlines g_0, g_1 are now:

$$\begin{aligned} span^0_{\text{post}} &:= \{((y_0, y_1), \{A\})\} \\ span^1_{\text{post}} &:= \{((y_0, y_1), \{C\})\}\,. \end{aligned}$$

Algorithm 15: CollapseGridlines

Input: The set of gridlines $\mathcal{G}_{x/y}$

begin
  foreach $g_i \in \mathcal{G}_{x/y}$ **do**
    $span^i_{\text{pre}} \leftarrow$ collect occupied spans and attached shapes
  end
  $c_{\text{pre}}(i,j) \leftarrow \mathit{false}\ \forall i,j \in [0, |\mathcal{G}_{x/y}| - 1]$
  $c_{\text{pre}}(i,j) \leftarrow \mathit{true}$ for each $g_i, g_j \in \mathcal{G}_{x/y}$ with a distance constraint
  calculate directed transitive closure of c_{pre}
  perform action on shapes
  foreach $g_i \in \mathcal{G}_{x/y}$ **do**
    collect $span^i_{\text{post}}$
    $collapse_i \leftarrow \emptyset$
    foreach *interval* (a,b) *in* $span^i_{pre}$ *or* $span^i_{post}$ *with attached shapes* S_{pre} *and* S_{post} *respectively* **do**
      if $S_{post} = \emptyset$ **then**
        $collapse_i \leftarrow collapse_i \cup ((a,b), \mathsf{empty})$
      else if $S_{pre} \subset S_{post}$ **then**
        $collapse_i \leftarrow collapse_i \cup ((a,b), \mathsf{collapse})$
      else
        $collapse_i \leftarrow collapse_i \cup ((a,b), \mathsf{prevent})$
      end
    end
  end
  collect $c_{\text{post}}(i,j) \leftarrow \mathit{true}$ for each pair $g_i, g_j \in \mathcal{G}_{x/y}$ as above
  calculate directed transitive closure of c_{post}
  foreach $g_i, g_j \in \mathcal{G}_{x/y} : g_i < g_j$ **do**
    foreach *interval* (a,b) *in* $collapse_i$ *or* $collapse_j$ *with collapse flags* c_i *and* c_j **do**
      if $(c_i = \mathsf{prevent} \wedge c_j \neq \mathsf{empty}) \vee (c_j = \mathsf{prevent} \wedge c_i \neq \mathsf{empty})$ **then** $c_{\text{post}}(i,j) \leftarrow \mathit{true}$
    end
  end
  foreach i,j *such that* $\neg c_{post}(i,j) \wedge c_{pre}(i,j)$ **do**
    TryMergeInto (g_i, g_j) and update transitive closure of c_{post}
  end
end

And for Fig. 4.18b the spans are:

$$span^0_{\text{pre}} := \{((y_0, y_3), \{A\})\}$$
$$span^1_{\text{pre}} := \emptyset$$
$$span^2_{\text{pre}} := \{((y_0, y_3), \{C\})\}\,.$$

For each gridline g_i $span^i_{\text{pre}}$ and $span^i_{\text{post}}$ are compared interval by interval in lines 13-21. More precisely, for each interval (a, b) the shapes S^i_{pre}, S^i_{post} attached to that interval in $span^i_{\text{pre}}$ and $span^i_{\text{post}}$ are compared.

For the first example this is straightforward. The spans of both gridlines g_0, g_1 only contain a single interval (y_0, y_1). $S^0_{\text{post}} = \{A\} \subset S^0_{\text{pre}} = \{A, B\}$, $S^1_{\text{post}} = \{C\} \subset S^1_{\text{pre}} = \{B, C\}$. Thus the interval (y_0, y_1) is marked as *collapse* on both gridlines g_0 and g_1.

For the second example the resulting collapse spans are

$$collapse_0 := \{((y_0, y_1), prevent), ((y_1, y_2), collapse), ((y_2, y_3), prevent)\}$$
$$collapse_1 := \{((y_1, y_2), empty)\}$$
$$collapse_2 := \{((y_0, y_1), prevent), ((y_1, y_2), collapse), ((y_2, y_3), prevent)\}$$

On gridline g_0 and g_2 intervals (y_0, y_1) and (y_2, y_3) remain unchanged and, by default, are marked as prevent. Only interval (y_1, y_2) is marked as collapsable. Gridline g_1 is empty after the deletion of B_0 and B_1.

Now the set of constraints c_{post} is collected. After B has been removed, $c_{\text{post}}(0, 1) = false$. After B_0 and B_1 have been removed from the second example, $c_{\text{post}}(0, 1) = c_{\text{post}}(1, 2) = false$.

The *collapse* interval sets are compared pairwise in lines 24–30. In the first example, the loop is without effect and c_{post} remains unchanged. In the second example however, the loop will find that gridlines g_0 and g_2 both marked intervals (y_0, y_1) and (y_2, y_3) with *prevent*, meaning on these intervals the attached shapes have not changed. Thus, these pairs of gridlines should remain separated and c_{post} is set to *true*.

For each pair of gridlines g_i, g_j that were separate before the shape manipulation, i.e., $c_{\text{pre}}(i, j) = true$, and that are not separated anymore, i.e., $c_{\text{post}}(i, j) = false$, algorithm TryMergeInto is called and the closure is updated accordingly. In the first example, gridlines g_0 and g_1 will thus be merged. In the second example, gridlines g_0 and g_2 will not be merged.

4.5 Summary

This chapter presented a set of elementary operations to construct instances of the Layout Problem, i.e., an arrangement of shapes and their attached

gridlines on a page. The algorithms described the consistent insertion of gridlines (Alg. 5), the deletion of shapes and subsequent collapsing of gridlines (Alg. 15), the merging of gridlines (Alg. 6 and Alg. 11), and the splitting of gridlines and their adjacent shapes (Alg. 7). These operations can be combined to build more complex operations. To implement an operation that aligns a set of shapes at their left gridlines, procedure TryMergeInto (Alg. 11) must be called first on the set of left gridlines of every shape. Afterwards, all left gridlines that could be moved to the same position can be merged (Alg. 6).

These elementary operations can also be used to implement a direct manipulation user interface. This chapter explained several important design decisions that were made during the development of the ICBM user interface. Previous constraint-based graphics applications used constraints to guide a layout algorithm and to limit subsequent user interactions. In contrast, the user interface algorithms in this chapter proposed much weaker constraint semantics. Since all constraints have been established by the user, they can be overridden by user interactions. Constraints only limit the solution space of the layout algorithm, but do not restrict the user himself. By dragging a shape across the page, the user effectively overrides all previously defined alignment relations between the dragged shapes and the remaining shapes. The weaker constraint semantics also become obvious when the user aligns several shapes. In this case, all gridlines at the same position are merged

The ICBM user interfaces fulfills Shneiderman's principles of direct manipulation interfaces detailed in 2.2.4. In the ICBM system, users interact directly with shapes and their constraints. When moved, shapes, i.e., the gridlines they are attached to, snap to each other to facilitate the establishment of alignment constraints. Both, the manipulated shapes and the alignment constraints are continuously visualized. Every action can easily be reversed manually or automatically via the *undo* and *redo* functions. The available constraints represent simple concepts familiar to users of common drawing and document layout applications.

Chapter 5

Constraint Solving

The previous chapter described how an instance of the Layout Problem $L = (P, \mathcal{C}, o)$ defined on page 44 is assembled. The user has created the page $L.P$ consisting of a set of shapes and their attached gridlines. The user has defined the constraint sets $C_{Min}, C_{Max}, C_{Equal}, C_{Merge} \in L.\mathcal{C}$. Several subproblems remain:

1. Complete constraint set $L.\mathcal{C}$:
   - For each shape that contains text, the shape must be large enough to fit the text. Text can only wrap at discrete locations and thus there is a discrete number of minimal width-height-configurations for shape containing text.
   - The layout algorithm has to derive additional constraints that define the shape positions and sizes relative to each other.
2. Define objective function $L.o$ such that
   - The layout degrades gracefully if L is over-constrained.
   - Conversely, if L is under-constrained, the solution layout has to be close to the user's expectactions.

5.1 Text Size Approximation

In the classic line break problem, the page width and word sizes are given and a wrapping of words into lines is sought that minimizes the overall deviation of all line widths from the page width.

Definition 18. The **line break optimization problem** is a triple $LB = (W, x, o)$ where $W = (w_0, \ldots, w_n), w_i \in \mathbb{R}$ is a list of word widths and $x \in \mathbb{R}$ is the available page width. The solution space to $\Delta(LB)$ is the set

$$\Delta(LB) = \{(i_0, \ldots, i_m) \,|\, 0 < i_0 < i_1 < \cdots < i_m = n \\ \wedge \forall 0 \leq j \leq m \,:\, \Sigma_{i_{j-1} < k \leq i_j} w_k \leq x \text{ with } i_{-1} := -1\}$$

where each i_j indicates a line break inserted after word w_j. Then, the objective function is a function $o \,:\, \Delta(LB) \times \mathbb{R} \to \mathbb{R}$ that defines a measure of quality for any pair of line break $l \in \Delta(LB)$ and page width x and the optimal solution to LB is defined as

$$\textbf{min}\, o(l, LB.x) \quad l \in \Delta(LB).$$

Typically, function o will compute the overall deviation of line widths from page width

$$o((i_0, \ldots, i_m), x) := \Sigma_{0 \leq j \leq m} \left(x - \Sigma_{i_{j-1} < k \leq i_j} w_k\right)^2$$

This problem has been analyzed and solved by Knuth and Plass in their 1981 paper "Breaking Paragraphs into Lines" [81]. For a text of n words, an optimal solution can be found using dynamic programming in $O(n^2)$ time.

The inverse of the line break optimization problem is how to compute the optimal text width given a list of word length. Let $L_i(W)$ be set of all possible text widths resulting from breaking a list of words W into i lines.

Definition 19. The **minimal text size problem** is, given a list of word widths $W = (w_0, \ldots, w_n)$ and assuming uniform word height, to compute the set of minimal width-height-configurations

$$WH = \{(w_i, h_i) \,|\, 0 \leq i \leq n,\ w_i = min_{w' \in L_i(W)}(w'),\ h_i = i + 1\}.$$

This problem has received some attention in recent years because it can be an important bottleneck when calculating table layouts, e.g., as part of an HTML renderer. Given all width-height-configurations for each cell of a table, the problem of choosing the best configuration for each cell such that the table in total has minimum height is a combinatorial optimization problem with exponential running time [115, 3]. Instead of solving this combinatorial problem, width-height-configurations are often approximated. Hurst et al. [67] approximated the size of text cells with the area covered by the content. The constant area constraint is hyperbolic and can be solved by a generalized form of quadratic programming called conic programming.

Anderson et al. [3] have developed a rather complex algorithm computing all width-height-configurations for text with uniform height that, according to the authors, has a running time of $O^*(n^{3/2})$ ignoring logarithmic terms.

5.1.1 Enumerating Text Sizes

Since the text size constraints need to be integrated into a larger constraint set, the flexibility and availability of powerful solvers were deciding factors in favor of a linear approximation for the ICBM layout system. Algorithm 16 solves the minimal text size problem for text with uniform line height in $O(n^2)$ using dynamic programming.

Algorithm 16: Enumerating all minimal width-height-configurations

Input: A sequence of words $X := x_1, \ldots, x_m$ of uniform height H
Output: A sequence of pairs $S := (w_i, h_i)$ such that w_i is the minimum width if the text has size h_i

```
1  begin
2      S ← ∅
3      foreach word x_j ∈ X do
4          W_1[j] ← W_1[j − 1] + width(x_j)
5      end
6      S ← S ∪ (W_1[m], H)
7      foreach line ∈ (2, m) do
8          prev_last_word ← line − 1
9          foreach last_word ∈ (line, m) do
10             w_Best ← max(W_{line−1}[prev_last_word],
                            W_1[last_word] − W_1[prev_last_word])
11             i ← prev_last_word + 1
12             while i < last_word do
13                 w ← max(W_{line−1}[i], W_1[last_word] − W_1[i])
14                 if w > w_Best then break
15                 w_Best ← w
16                 prev_last_word ← i
17             end
18             W_line[last_word] ← w_Best
19         end
20         S ← S ∪ (W_line[m], H · line)
21     end
22 end
```

The algorithm calculates, for every possible number of lines, the minimum needed text width. The algorithm maintains the invariant that $W_k[j]$ is the minimum text width that fits the words $x_0, \ldots, x_j$ on k lines. The algorithm accumulates the total width of $x_1, \ldots, x_j$ in variable $W_1[j]$. If all words are

drawn on a single line, their total width is $W_1[m]$ and their height is H. The algorithm loops over every possible number of lines and calculates the minimum text width recursively. If all m words must be placed on k lines, line $k-1$ will end with word $x, 1 \leq x < m$. The width of all previous $k-1$ lines is given by $W_{k-1}[x]$. Line k has width $W_1[m] - W_1[x]$. The problem is finding an x that minimizes $\max(W_{k-1}[x], W_1[m] - W_1[x])$.

The outer loop at line 11 iterates over the current number of lines *line*. Variable *prev_last_word* is the index of the last word on line *line* − 1. Variable *last_word* is the index of the last word on line *line*. The loop over *last_word* in line 9 fills the dynamic programming tableau, calculating in each iteration a value $W_{line}[last\_word]$. Every time *last_word* is incremented, i.e., another word is added to line *line*, the width of the last line $W_1[last\_word] - W_1[prev\_last\_word]$ could become so large, that the total text width would decrease, if more words were added to the previous *line*−1 lines, i.e., if *prev_last_word* were incremented. This is checked in the while loop at line 12. The smallest width is kept in variable w_{Best}.

Algorithm 17 describes how the width-height-configurations of individual paragraphs can be combined into the width-height-configurations of minimal width for a multi-paragraph text. For every paragraph P_i, the algorithm computes the width-height-configurations S_i using Alg. 16. These are ordered by increasing width. Variable W keeps track of the narrowest possible width that fits all paragraphs and H is their combined height at the same width. Every pair (w_j, h_j) in set S_i is transformed so that it contains the next largest width and the associated decrease in height if the text were drawn with the wider width.

Obviously, all paragraphs fit into a box of width W and height H. By construction, the text cannot fit into a box narrower than W. Every pair (w_j, h_j) means there is a paragraph whose height decreases by h_j if the total width reaches w_j. In the first iteration, $w_j > W$ and therefore (W, H) will be added to S. w_j is the narrowest possible text width larger than W and W is set to w_j. If $\{(w_i, h_i) \mid w_i = W\}$ is the set of all width-height pairs of width W, the total text height H decreases by $\sum_i h_i$. After the loop has iterated over all such pairs (w_i, h_i), S' does not contain anymore elements, or the next element $(w_k, h_k$ has again width $w_k > W$.

5.1.2 Approximating Width-Height-Configurations

The minimal area width-height configurations are approximated with linear functions minimizing the maximum distance to the step-wise width-height function. Figure 5.1 shows the possible sizes for the displayed text as dashed rectangles and the linear functions approximating the width-height pairs.

Algorithm 17: Combining width-height-configurations of individual paragraphs

Input: A sequence of paragraphs $P := \{P_1, \ldots, P_n\}$ with each P_i consisting of words $x_1^i, \ldots, x_m^i$

Output: A sequence of pairs $S := (w_i, h_i)$ such that all paragraphs P fit into text boxes of size (w_i, h_i)

```
1  begin
2      (W, H) ← (0, 0)
3      S' ← ∅
4      foreach P_i ∈ P do
5          S_i ← CalculateParagraphSizes(P_i)
6          sort S_i = {(w_1, h_1), ... (w_m, h_m)} by w_j in ascending order
7          (W, H) ← (max(W, w_1), H + h_1)
8          foreach (w_j, h_j) ∈ S_i do  (w_j, h_j) ← (w_{j+1}, h_j − h_{j+1})
9          S' ← S' ∪ S_i
10     end
11     sort (w_j, h_j) ∈ S' by w_j in ascending order
12     S ← ∅ is the set of width-height pairs of all paragraphs combined
13     foreach (w_j, h_j) ∈ S' do
14         if w_j > W then  S ← S ∪ (W, H),  W ← w_j
15         H ← H − h_j
16     end
17     S ← S ∪ (W, H)
18 end
```

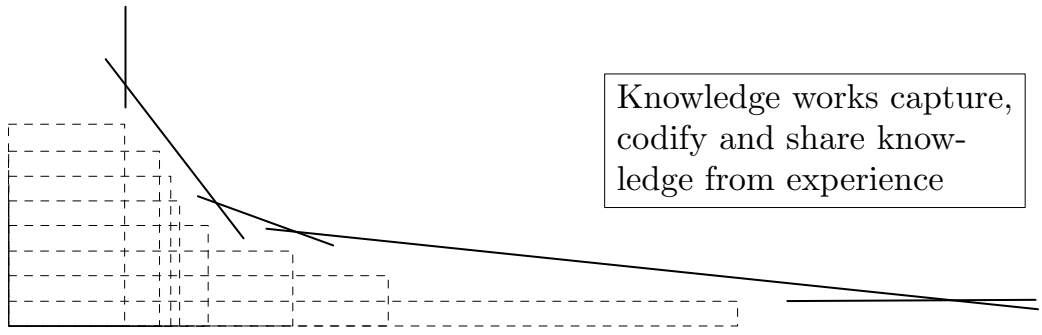


Figure 5.1: Finding a linear approximation of the width-height configurations of text content.

Algorithm 18: Linear approximation of width-height-configurations

Input: A set of width-height pairs $P = \{(w_1, h_1), \ldots, (w_n, h_n)\}$
Output: Linear inequalities $L = \{m_i \cdot w + n_i \leq h\}$ giving lower bound for text box height h depending on width w, lower bound w_{min} for width w.

begin
  foreach $k = 1 \ldots n-1$ **do**
    $pivot \leftarrow (w_{k+1}, h_k)$
    $C \leftarrow \emptyset$
    foreach $(w_i, h_i) \in P$ **do**
      $\vec{v} \leftarrow (w_i, h_i) - pivot$
      if $\vec{v}.x \geq 0$ **then**
        $C \leftarrow C \cup \{f(x) = |\vec{v}|x - |\vec{v}| \cdot \text{atan2}(\vec{v})\}$
      else
        $C \leftarrow C \cup \{f(x) = -|\vec{v}|x + |\vec{v}| \cdot (\text{atan2}(\vec{v}) - \pi)\}$
      end
    end
    RemoveRedundant(C)
    sort $C = \{f_i(x) = m_i x + n_i\}$ such that $f_i < f_j \leftrightarrow m_i < m_j$
    calculate set P of intersections between each pair $f_{i-1}, f_i \in C$
    binary search over P to find $p_i \in P$ with $p_i.y = \min$
    $m_k \leftarrow \tan(p_i.x)$
    $n_k \leftarrow pivot.y - pivot.x \cdot m_k$
    $L \leftarrow L \cup \{m_k \cdot w + n_k \leq h\}$
  end
  $L \leftarrow L \cup \{0 \cdot w + h_n \leq h\}$
  RemoveRedundant (L)
  $w_{min} \leftarrow w_1$
  delete constraints from L redundant considering $w \geq w_{min}$
end

The algorithm makes several simplifying assumptions to find a satisfactory linear approximation in $\mathcal{O}(n^2)$ given a list of n width-height-configurations. First, the algorithm generates one linear function for each point *pivot* set to a corner (w_{i+1}, h_i), i.e., to the point that guarantees the necessary height h_i for a rectangle up to width w_{i+1} as illustrated in Fig. 5.1. Second, for each point *pivot* this function is found by minimizing the arc length between the function and every other point (w_j, h_j) and not the distance between the linear function and same point.

The inner loop of lines 5-11 finds the optimal slope for the linear function going through *pivot*. In each iteration, vector $\vec{v}$ is the vector between point (w_i, h_i) and *pivot* as illustrated in Fig. 5.2. Function *atan2* returns the angle between $\vec{v}$ and the x-plane. The length of an arc of angle x' with radius $|\vec{v}|$ is $2\pi|\vec{v}| \cdot x'$. If we want to express the arc angle relative to the x-plane the equation becomes $2\pi|\vec{v}| \cdot (x - \text{atan2}(\vec{v}))$ for $\vec{v}.x \geq 0$ or omitting the constant part 2π we can write

$$f(x) = |\vec{v}| \cdot (x - \text{atan2}(\vec{v})) = |\vec{v}|x - |\vec{v}|\text{atan2}(\vec{v}).$$

Each such function f calculates the arc length (divided by 2π) from a linear function of angle x going through *pivot* to a single point (w_i, h_i). By ordering these functions by slope in ascending order, the optimal angle x that minimizes $f(x)$ can be found using binary search in logarithmic time. The functions $f(x)$ ordered by ascending slope form the boundary of a convex polytope that must take its minimum value in one of its extremal points, i.e., in one of the intersections calculated in line 15.

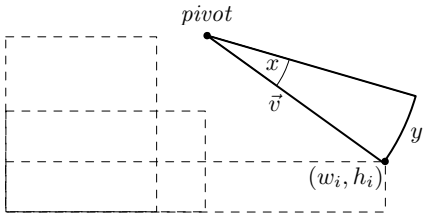


Figure 5.2: Finding a linear approximation of the width-height configurations of text content.

Algorithm 18 has made use of a function *RemoveRedundant* that as the name implies removes all redundant linear constraints from the input set. Its straightforward implementation is shown in Alg. 19. All linear equations are sorted by slope and by offset n. Set P is the set of intersection points between subsequent functions f_{i-1}, f_i which is updated in parallel and serves

to identify redundant constraints in line 7. For each new function f_i the algorithm looks backwards at set F' of the already added functions. The points in P define the extremal points of the convex set defined by F'. The test in line 7 checks if the next function f_i restricts the convex set in such a way that equations $f_j \dots f_k$ are themselves no longer restricting the convex set and are thus redundant.

Algorithm 19: RemoveRedundant

Input: A set of linear functions $F := \{f_1, \dots, f_n \mid f_i(x) := m_i x + n_i\}$.
Output: A sorted set $F' \subseteq F$ without redundant functions

```
1  begin
2      sort F such that f_i < f_j ↔ m_i < m_j ∨ (m_i = m_j ∧ n_i > n_j)
3      F' ← {f_1}
4      P ← {(0, ∞)}
5      foreach i = 2 ... n do
6          if m_{i-1} < m_i then
7              if ∃j ∈ (1, i-1) : f_j ∈ F', p_j ∈ P ∧ f_i(p_j.x) ≥ p_j.y then
8                  remove all redundant f_k ∈ F', k ≥ j
9              end
10             F' ← F' ∪ {f_i}
11             p_i.x ← (n_{i-1} − n_i)/(m_i − m_{i-1})
12             p_i.y ← f_i(p_i.x)
13             P ← P ∪ {p_i}
14         end
15     end
16 end
```

These linear constraints complete the constraint set $C_{\text{Min}} \in L.\mathcal{C}$ of a layout problem L. The linear approximation of discrete text sizes is quite loose as Fig. 5.2 illustrates. In a layout satisfying this approximation of text size constraints, each text box will have a size (w, h) such that there is a minimum text size (w', h') as computed by Alg. 16 with $w \geq w' \wedge h \geq h'$. In most cases, even $w > w' \wedge h > h'$ will hold. Thus, a layout solution may be improved by a second run of the layout algorithm that replaces the linear approximation of text size constraints with a single constraint that the text width be at least w' and the height at least h'. The resulting layout could be more compact if the stricter constraint is also satisfiable.

5.2 Layout Constraints

So far, the constraints capturing properties of the layout, i.e., the spatial relations between different shapes, are still missing. If two shapes are adjacent, there is a gap of free space between the two shapes. As long as the size of this gap is larger than zero, the adjacency relation is not violated. When a gap constraint between two gridlines is inserted, both gridlines are forced apart by the solver and additionally, the order of the two gridlines will be fixed. It is thus important to consider the question where gap constraints must be inserted carefully.

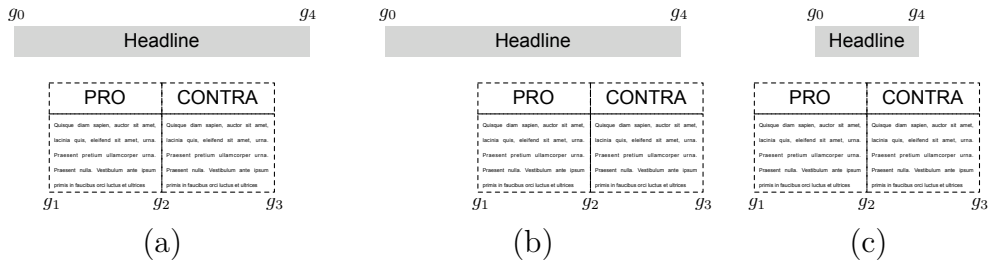


Figure 5.3: The horizontal position of the heading should be left unconstrained.

In a naive implementation, gap constraints could be inserted between each pair $g_i, g_j \in \mathcal{G}_{x/y}$ with $g_i.v < g_j.v$. This would preserve the total gridline order defined by the user interaction. However, the result is a system that feels overly rigid as the scenario in Fig. 5.3 shows. If the user has drawn the first example of Fig. 5.3a, gap constraints will be inserted between pairs (g_0, g_1), (g_1, g_2), (g_2, g_3), and (g_3, g_4). The layout that centers the headline above the table is clearly the solution that is probably intended. If the user adds more content left of the table, the headline should remain centered above both the added content and the table. Yet, with the set of gap constraints added before, a solution like the one of Fig. 5.3b is not possible. Gridlines g_3 and g_4 have changed their order, which the gap constraints would have prevented. Conversely, if the user did not anticipate the total amount of text she needed for the headline and simply created a situation as in Fig. 5.3c, the result of Fig. 5.3a becomes unattainable. Gap constraints limit the position of gridline g_0 to be between g_1 and g_2. The same holds for g_4, g_2 and g_3. If more text is added to the headline, the table becomes wider too. Clearly, this is not a desirable behavior. The horizontal gridlines of the table and the headline should not constrain each other. In vertical direction, the problem is different. The headline must not move below the table.

The difference between the two cases is that the headline can be moved horizontally without colliding with the table. This is impossible in vertical direction. Gap constraints should be inserted in vertical direction to keep the headline above the table. The bottom gridline of the headline is occupied between gridlines g_0, g_4, the top of the table is occupied between gridlines g_1, g_3 in Fig. 5.3a. Both gridline intervals overlap and a gap constraint should be inserted that keeps the bottom of the headline above the top of the table.

Such a formulation leads to an under-constrained problem if the user inserts two shapes diagonal to each other as in Fig. 5.4. No gap constraints are inserted between the table and the headline. The user has not specified if the headline should remain to the top right of the table, or if it is allowed to move horizontally or vertically. Instead of resolving this ambiguity in some arbitrary way, the relative position of the headline to the table is left unconstrained. The shape positions could be defined through constraints with other shapes. Even if both positions are left unconstrained, there is no problem unless both shapes intersect in the solution layout. A post-processing step that is explained in section 5.3, resolves the ambiguity only when necessary, i.e., if both shapes indeed intersect after the constraint solver has run.

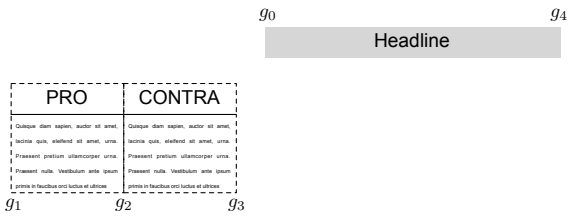


Figure 5.4: No gap constraints are inserted at all between shapes diagonal to each other.

The algorithm that adds the gap constraints is shown in detail in Alg. 20. First, in line 2, the span of each gridline is computed and in the next line the array *Gap* is initialized that will be used to keep track of the inserted gap constraints. The algorithm proceeds by moving index *high* forward from 1 up to and including $n + 1$, i.e., one index after the last gridline. From the gridline with index *high*, the algorithm goes backwards using index *low* in the inner loop of line 6. The algorithm then checks if the spans of gridlines *high* and *low* overlap and inserts a gap constraint between both gridlines.

Two special cases deal with the very first and the very last gridline. If a gridline is outside of the page boundaries, its value is only bounded on one side by the page. Constraints are added that will keep these gridlines as

Algorithm 20: Add Gap Constraints

Input: Gridlines $\mathcal{G}_{x/y} = \{g_1, \ldots, g_n\}$, page extent $d \in \mathbb{R}$
Output: The set of gap constraints C_{Gap} and the set of constraints of objects outside the page C_{Close}

```
1  begin
2      Span ← {span_1, ..., span_n | span_i = span(g_i), g_i ∈ G_{x/y}}
3      Gap(i, j) ← false ∀i, j ∈ (0, n+1), i < j // array of all pairs i, j
4      foreach high = 1 ... n+1 do
5          if high = n+1 ∨ span_high ≠ ∅ then
6              foreach low = high − 1 ... 0 do
7                  if ¬Gap(low, high) ∧ (low = 0 ∨ span_low ≠ ∅) then
8                      add ← false
9                      if low = 0 then
10                         add ← true
11                         C_Close ← C_Close ∪ {g_high.v − d/2 ≥ z, z ≤ 0}
12                     else if high = n+1 then
13                         add ← true
14                         C_Close ← C_Close ∪ {−g_low.v + d/2 ≥ z, z ≤ 0}
15                     else if g_low.v < g_high.v∧ span_low ∩ span_high ≠ ∅ then
16                         add ← true
17                         if g_high, g_low in one partition in P_Merge then
18                             C_Gap ← C_Gap ∪ {g_high.v − g_low.v ≥ 0}
19                         else
20                             C_Gap ← C_Gap ∪ {g_high.v − g_low.v > 0}
21                         end
22                     end
23                     if add then
24                         foreach 0 ≤ i < low do
25                             Gap(i, high) ← Gap(i, low) ∨ Gap(i, high)
26                         end
27                         Gap(low, high) ← true
28                     end
29                 end
30             end
31         end
32     end
33 end
```

close as possible to the page. If $low = 0$, then the algorithm adds a constraint keeping gridline *high* close to the page. The same happens for gridline *low*, if $high = n + 1$. In both cases, a constraint containing a variable $z \leq 0$ is inserted, where z bounds the distance of g_{low} or g_{high} from the page center. If z is included in the objective function, then the gridlines can be positioned as close as possible to the page.

The general case is treated in line 15. The spans at index *low* and *high* are both not empty, neither is their intersection and the gridline at index *low* has a lower value than gridline *high*. The gap constraint keeps both gridlines separated. If both gridlines are marked in C_{Merge} as meant to be merged (see Section 28), the gap constraint allows for equality with zero. It is worth repeating at this point, that there are four gridlines that always exist and that lie on the page boundary. If no other adjacent shape exists, there will at the very least be gap constraints between a shape and the page boundary. Finally, if any constraint between gridlines *low* and *high* has been added, variable *add* is true and *Gap(low, high)* is set to true. The gap relation is transitive, i.e., for $i < low < high : Gap(i, low) \wedge Gap(low,\ high) \rightarrow Gap(i, high)$.

5.2.1 Objective Function

The constraints formulated so far more or less followed naturally from the shapes' content or from the arrangement of shapes on the page. The solution to the layout problem however and the desired properties of a layout solution are encoded in the objective function. Some authors have tried to define properties of "nice" layouts [54] or have even tried to develop algorithms based on such measures [85]. Marriott et al. mentioned in their recent work on document layout [92] that users accepted automatically generated layouts if they could not be improved locally.

What "nice-ness" means in regard to the layout of a single shape should be expressed by the shape's width and height constraints. The layout algorithm should favor regular and predictable layout solutions to increase the user's acceptance of the generated results. This means essentially, that the available space on a page should be distributed evenly between the gaps in horizontal and vertical direction. Conversely, if the layout is over-constrained and no feasible solution to a strict set of constraints exists, the layout algorithm should spread the error evenly between the constraints. The goal is that the automatically generated solutions degrade gracefully with tightening constraints, always yielding a solution that is as good as possible while simultaneously providing a visual cue that the constraint set is over-constrained.

Maximizing Constraint Satisfaction

The layout problem resembles a constrained resource allocation problem in that a limited resource, i.e., page space, is assigned to different gaps subject to a set of distance constraints. Some gaps have minimum sizes, others have maximum sizes or both. To solve such resource allocation and planning problems, the so-called goal programming approach has been developed. Where linear programming problems maximize a single objective, goal programming problems minimize the deviation from a set of specified objectives [82, 37, 106, 114, 23].

Definition 20. A **weighted goal programming problem** WGP is the problem of finding

$$\begin{aligned} \textbf{min } z = \sum_{i=1}^{n} (u_i n_i + v_i p_i) \quad \text{subject to} \\ f_i(x) + n_i - p_i = b_i \quad i = 1 \ldots n \\ Ax = b \end{aligned}$$

where each $f_i : \mathbb{R}^n \to \mathbb{R}$ is a goal function, $b_i \in \mathbb{R}$ is the desired goal value for $f_i(x)$, and the $n_i, p_i \in \mathbb{R}$ are the negative and positive deviation of $f_i(x)$ from this goal value. Given weights $u_i, v_i \in \mathbb{R}$ the problem is thus to minimize the weighted deviation over all i. As in linear programming, $Ax = b$ is an optional set of hard constraints.

Constraints with negative or positive error terms are usually called *soft constraints* and they have been used, e.g., in the Auckland Layout Manager developed by Lutteroth et al. [86]. The Cassowary constraint solver [7] solves hierarchical sets of linear constraints by minimizing the weighted sum of positive and negative constraint errors incurred on non-required constraints.

Given an instance of the layout problem $L = (P, \mathcal{C}, o)$, each constraint set in $\mathcal{C}$ can be transformed into a set of soft constraints by adding a so-called *scale* variable. The scale variable is a negative error term that represents the amount by which the constraint equation deviates from its target value. It is a negative error term because the constraint equations never exceed their target value. Then, the objective function $L.o$ can be defined in terms of these *scale* variables.

The idea of a scaled constraint can most easily be applied to the set C_{Min} of minimum distance constraints. Each constraint in $e_k \in C_{\text{Min}}$ takes the form

$$e_k : \sum_{i=1}^{n} c_i \cdot g_i.v \geq d \qquad \neg g_i.f, g_i \in \mathcal{G}_x \cup \mathcal{G}_y, c_i, d \in \mathbb{R}, d \geq 0$$

The constraint set C'_{Min} is defined by adding a *scale* variable $s_k \in \mathbb{R}$ to every constraint equation $e_k \in C_{\text{Min}}$. The resulting constraints have the form

$$e'_k \; : \; \sum_{i=1}^{n} c_i \cdot g_i.v \geq d \cdot s_k.$$

Definition 21. A **scaled constraint set** $SC = (S, H, u)$ is a triple consisting an of an upper bound $u \in \mathbb{R}$, a set of scaled constraint equations

$$S = \left\{ \sum_{i=1}^{n} c_i \cdot g_i.v \geq d \cdot s_k \,|\, \neg g_i.f, g_i \in \mathcal{G}_x \cup \mathcal{G}_y, \;\; c_i, d, s_k \in \mathbb{R}, s_k \leq u \right\},$$

where each scale variable s_k is only part of a single constraint equation $e_k \in S$, and a possibly empty set of hard constraints H.

Given a scaled constraint set

$$SC_{Min} = (C'_{Min}, \emptyset, u := 1)$$

there always exists an n-tuple of scale variables such that all scaled constraint equations in C'_{Min} are satisfied since in every constraint $d \geq 0$ and the s_k can be made arbitrarily small until every inequality is satisfied. If $\forall k \; : \; s_k = 1$, all minimum distance constraints in C_{Min} are satisfied.

Formulating Scaled Constraint Sets

The constraint sets $C_{\text{Max}}, C_{\text{Equal}}, C_{\text{Gap}}$ and C_{Close} have to be transformed into their scaled equivalents:

1. The set of **Maximum Distance** constraints

   $$C = \left\{ \sum_{i=1}^{n} c_i \cdot g_i.v \leq d \,|\, \neg g_i.f, g_i \in \mathcal{G}_x \cup \mathcal{G}_y, \;\; c_i, d \in \mathbb{R}, \;\; d \geq 0 \right\},$$

   is transformed into

   $$C' = \left\{ \sum_{i=1}^{n} -c_i \cdot g_i.v \geq -d \cdot s_k \,|\, \neg g_i.f, g_i \in \mathcal{G}_x \cup \mathcal{G}_y, \;\; c_i, d \in \mathbb{R}, \;\; d \geq 0 \right\},$$

   and the scaled constraint set is defined as $SC_{Max} = (C', \emptyset, -1)$.

2. **Equal Distance**

   With gridlines $g_i, g^j \in \mathcal{G}_x \cup \mathcal{G}_y$ and factors $c_i, c_j \in \mathbb{R}$, equal distance constraints have the form

   $$\sum_{i=1}^{n} c_i \cdot g_i.v = \cdots = \sum_{j=1}^{m} c_j \cdot g_j.v.$$

By introducing a common variable $z_k \in \mathbb{R}$, the single equation can be split into several constraints

$$\sum_{i=1}^{n} c_i \cdot g_i.v = z_k \wedge \cdots \wedge \sum_{j=1}^{m} c_j \cdot g_j.v = z_k.$$

Each individual constraint can be further transformed into

$$\sum_{i=1}^{n} c_i \cdot g_i.v - z_k \leq 0 \wedge \sum_{i=1}^{n} c_i \cdot g_i.v - z_k \geq s_k.$$

All left equations form the set of hard constraints H, the right equations the set of scaled constraints S, and $SC_{Equal} = (S, H, 0)$.

3. **Gap Constraints**

   For every gap constraints

   $$g_r - g_l > 0$$

   the scaled constraint

   $$g_r - g_l \geq s_k$$

   is added to set S', and the unscaled hard constraint $g_r - g_l > 0$ is added to H'. For gap constraints $g_r - g_l \geq 0$ only the scaled form is added to S'. $SC_{Gap} = (S', H', \infty)$.

4. **Close To Slide Constraints**

   The last remaining constraints are those inserted in Alg. 20 to keep gridlines close to the page. The constraint is already in scaled form:

   $$g_{high} - d/2 \geq s_k \qquad \text{or} \quad -g_{low} + d/2 \geq s_k$$

   Maximizing variable s_k with $s_k \leq 0$ will move gridlines g_{high} and g_{low} as close to the page center as possible. The constraint is only inserted for gridlines to the left, right, top or the bottom of the page. The upper bound for gridline g_{high} is thus the left or top gridline on the page boundary and the above constraint functions as a lower bound to the position of g_{high}. For g_{low} the right or bottom page boundary form the lower bound and the above constraint limits the upper bound. Therefore, $SC_{Close} = (C_{Close}, \emptyset, 0)$.

The Scaled Layout Problem

In all scaled constraint sets except SC_{Gap}, the scale variable is a negative error term that must be maximized up to the given upper bound. In gap constraints, the scale variable is the size of the gap itself which has no upper bound. Assuming not all constraints can be satisfied, how should the constraint error be distributed between the scale variables, or how should the available page space be distributed among the gap constraints? For example, if not all minimum distance constraints can be satisfied, should the number of unsatisfied constraint equations be minimized, the largest deviation of some scale variable s_k from its upper bound, or some other measure?

The scaled constraint sets can be ordered according to their priority from highest to lowest: $\mathcal{SC} = (SC_{Min}, SC_{Max}, SC_{Equal}, SC_{Gap}, SC_{Close})$. Minimum distance constraints guarantee the minimum shape sizes necessary to fit a shape's content. Gap constraints only distribute the remaining space once every shape has been assigned a minimum extent. The least important constraints belong to constraint set C_{Close}. Thus, it seems preferable to maximize the scale variables belonging to minimum size constraints first. For each scaled constraint set SC_i, $\vec{s}_i \in \mathbb{R}^{|SC_i.S|}$ is the vector of scale variables occurring in the soft constraints $SC_i.S$. Let

$$x := (\vec{s}_{\text{Min}}, \vec{s}_{\text{Max}}, \vec{s}_{\text{Equal}}, \vec{s}_{\text{Gap}}, \vec{s}_{\text{Close}})$$

be the vector composed of the scale variable vectors for each scaled constraint set. The problem of finding an optimal assignment of scale variables can be treated independently for each constraint set, i.e., for each hierarchy level. As previously described, the distribution of constraint errors and the distribution of gap sizes should be even and regular. The problem of equitable resource allocation has been studied extensively in the Operations Research literature. Space is a resource that is distributed among gaps and infeasibility is a resource that can be distributed over the scale variables.

For the set of minimum distance constraints, there could be two feasible solutions: A single minimum distance constraint may have a very small scale value, meaning that the deviation between the desired and the obtainable minimum distance is very large, or alternatively, many minimum distance constraints may have scale variables only a little less than one. The latter solution distributes the dissatisfaction more evenly among the constraints and is preferable to the first solution.

Several definitions of optimality are usually referred to in the context of goal-programming problems [43, 33], each having certain desirable properties.

Definition 22. A feasible solution $x = (x_0, \ldots, x_n) \in C$ is called *Pareto optimal*, if there is no other feasible solution $x' \in C$ whose components

are each not less than the components of x and there exists at least one component of x' strictly larger than that of x, i.e.,

$$\nexists x' \in C : \forall i : x'_i \geq x_i \wedge \exists j : x'_j > x_j \qquad i, j \in [0, n]$$

No feasible solution to the layout problem should be considered optimal if it does not meet the definition of Pareto-optimality. However, the set of Pareto-optimal solutions can still be very large.

Definition 23. A solution $x \in C$ to a maximization problem is called *min-ordering* optimal if the smallest component is maximal:

$$\forall x' \in C : \min_i(x'_i) \leq \min_j(x_j) \qquad i, j \in [0, n].$$

The value of a min-ordering optimal solution is uniquely defined but not every min-ordering solution is also Pareto-optimal. The min-ordering criterion does not restrict the values of vector components other than the one with the smallest value.

Definition 24. A vector $x \in C$ is called *lexicographically maximal* if

$$\forall x' \in C : \exists i : \forall j \in [1, i-1] : x'_j = x_j \wedge x'_i \leq x_i \qquad i \in [0, n].$$

Lexicographic optimality thus compares all components of a feasible solution and the optimal solution value is again uniquely defined. However, the definition of optimality requires an a-priori order on the members of a feasible solution vector x. In resource allocation problems, the components of x can frequently be ordered by priority, i.e., the activities requiring resources are not equally important. This does not apply to the layout problem. All scale variables are equally important.

The layout problem needs a definition of optimality that is stricter than all three previous optimality definitions and that requires no a priori information. Such a definition of optimality has first been described in a paper concerning the optimal strategy choice in a two player zero sum game without prior information [11] and it has since been used in many different applications contexts, e.g., bandwidth allocation and network planning [103, 101].

Definition 25. A vector $a := (a_1, \ldots, a_n)$ is called lexicographically larger than $b := (b_1, \ldots, b_n)$, $a, b \in \mathbb{R}^n$, denoted $a >_{\text{lex}} b$, if

$$\exists i \in [1, n] : \forall j \in [1, i-1] : a_j = b_j \wedge a_i > b_i$$

Definition 26. Given a vector $a \in \mathbb{R}^n$, vector $\langle a \rangle := (a_{\langle 1 \rangle}, \ldots, a_{\langle n \rangle})$ is formed from a by rearranging the components of a in ascending order, i.e.,

$$\forall i \in [1, n-1] : a_{\langle i \rangle} \leq a_{\langle i+1 \rangle}$$

Definition 27. A vector $a := (a_1, \ldots, a_n)$ is called lexicographically min-ordering larger than $b := (b_1, \ldots, b_n)$, $a, b \in \mathbb{R}^n$, denoted $a >_{\text{lex-mo}} b$, if

$$\langle a \rangle >_{lex} \langle b \rangle$$

Accordingly, for a set $X \subseteq \mathbb{R}^n$, element $x \in X$ is lex-mo maximal, denoted $\max_{\text{lex-mo}} x$, if and only if $\nexists y \in X : y >_{\text{lex-mo}} x$.

It follows that a lex-mo optimal solution is also a min-ordering optimal solution and a lex-mo optimal solution is always Pareto optimal [33].

The lexicographic min-ordering optimality criterion has a very intuitive meaning. The smallest scale variable is maximized, i.e., the least satisfiable constraint is as close to satisfaction as possible. Correspondingly, the smallest gap, i.e., the most constrained gap, is made as large as possible too, and the remaining gap space is further optimized. The second smallest scale variable is maximized, i.e., the second most constrained gap width is maximized as well and so on. The final lex-mo optimal solution value is again uniquely defined, i.e., all lex-mo optimal solutions x are equivalent in the sense that their permuted vectors $\langle x \rangle$ are the same.

Definition 28. Vector $x := (x_1, \ldots, x_n)$, with vector components $x_i \in \mathbb{R}^{m_i}$, is called component-wise lex-mo maximal, or c-$\max_{\text{lex-mo}} x$, if each x_i is lex-mo maximal:

$$\text{c-}\max_{\text{lex-mo}} x := (\max_{\text{lex-mo}} x_0, \ldots, \max_{\text{lex-mo}} x_n)\,.$$

Definition 29. The **Scaled Layout Problem** is a pair $SLP = (P, \mathcal{SC})$ where P is a page as of Def. 11 and

$$\mathcal{SC} = (SC_{Min}, SC_{Max}, SC_{Equal}, SC_{Gap}, SC_{Close})$$

is the list of scaled constraint sets defined above, in order of their priority. The problem's solution space is defined as

$$\Delta(SLP) = \mathbb{R}^{|SC_{Min}.S|+|SC_{Max}.S|+|SC_{Equal}.S|+|SC_{Gap}.S|+|SC_{Close}.S|}.$$

Any $x \in \Delta(SLP)$ can be written partitioned into its component scale variable vectors defined above as $x = (\vec{s}_{\text{Min}}, \vec{s}_{\text{Max}}, \vec{s}_{\text{Equal}}, \vec{s}_{\text{Gap}}, \vec{s}_{\text{Close}})$. The solution to a scaled layout problem is the vector

$$\begin{aligned} \text{c-}\max_{\text{lex-mo}} x \quad & x \in \Delta(SLP) \\ \text{subject to} \quad & \bigcup_{SC_i \in SLP.\mathcal{SC}} SC_i.S \\ \text{and} \quad & \bigcup_{SC_i \in SLP.\mathcal{SC}} SC_i.H \end{aligned}$$

5.2.2 Solving Lexicographic Min-Ordering Problems

Lexicographic maximization problems can be solved by solving a sequence of convex optimization problems [112, 103]. Given a convex set of feasible solutions X, and feasible solutions $x := \{x_1, \ldots, x_n\}$, the problem is finding

$$\begin{aligned} &\max_{\text{lex-mo}}\ x \\ &\quad x \in X \end{aligned} \tag{5.1}$$

By solving the optimization problem

$$\begin{aligned} &\max\ \tau \\ &\quad x_i \geq \tau, \forall i \in [1, n] \\ &\quad x \in X \end{aligned} \tag{5.2}$$

the maximum possible value of the smallest $x_i \in x$ can be found.

Given any optimal solution vector $[x \mid \tau]$ to the above problem (5.2), there must be at least one $x_j \in x$ such that the equation $x_j = \tau$ is satisfied. If no such x_j existed, there would be a $\tau' := \min_j(x_j) > \tau$ and $[x \mid \tau']$ would be the optimal solution. Let $J := \{j \mid x_j = \tau\}$.

For each $j \in J$, the auxiliary optimization problem

$$\begin{aligned} &\max\ x_j \\ &\quad x_i \geq \tau, \forall i \in [1, n], i \neq j \\ &\quad x \in X \end{aligned} \tag{5.3}$$

is solved and there must be at least one $j \in J$ such that $\max x_j = \tau$, i.e., at least one x_j whose value cannot be further increased without another x_i decreasing.

Proof. If this were not the case, there would be $|J|$ vectors $x^{j_1}, \ldots, x^{j_m} \in X$, $j_1, \ldots, j_m \in J$, such that for each x^{j_k} it holds that $\forall i \in [1, n]\ x_i^{j_k} \geq \tau$ and $x_{j_k}^{j_k} > \tau$. Then however, there must be a convex combination of x^{j_k} $x' := \sum_{j_k \in J} \lambda_{j_k} x^{j_k}$ with $\sum_{j \in J} \lambda_j = 1$. It follows that $\forall i \in [1, n]\ x'_i \geq \tau$ and $\forall j \in J\ x'_j > \tau$. Because X is convex, $x' \in X$. With the same argument, any convex combination of x' and the vector x from the optimal solution $[x \mid \tau]$ of problem (5.2) must be in X as well. For any λ, $0 < \lambda < 1$, the vector $x'' = \lambda x' + (1 - \lambda)x$ must be in X and for every such vector it follows that $\forall i \in [1, n]\ x''_i > \tau$. This contradicts the maximality of τ. □

Hence, there is a subset of indices $K \subseteq J$ whose corresponding vector components cannot be further increased. It follows that in every lex-mo maximal solution $x \in X$ to (5.1), $x_k = \tau$ for $k \in K$ and $x_i > \tau$ for $i \notin K$. It followed from (5.3) that in all feasible solutions $x' \in X$ with $x'_k > \tau$ for any $k \in K$ there must be a $j \in [1, n], j \notin K$ with $x_j < \tau$ and thus $x' <_{\text{lex-mo}} x$.

Setting $\tau_0 = \tau$, each constraint $x_k \geq \tau$ in (5.2) is replaced with $x_k \geq \tau_0$ and (5.2) can be solved again, yielding the next value τ. Let set $B \subseteq [1, n]$ be the set of indices for which a lower bound has been found already and let B' be the set of remaining indices. Furthermore, let T be the set of lower bounds for the vector components with indices in B. In the i-th iteration, the optimization problem is

$$\begin{aligned} \max\ & \tau \\ & x_i \geq \tau, \forall i \in B' \\ & x_j \geq t_j, \forall j \in B, t_j \in T \\ & x \in X \end{aligned} \tag{5.4}$$

5.2.3 Linear Lexicographic Min-Ordering Problems

The number auxiliary problems (5.3) that need to be solved until the lex-mo maximal solution is found can become extremely large. If the convex set of feasible solutions X is given by linear constraints, the solution to (5.1) can be found much more efficiently. A linear optimization problem in symmetric form

$$\begin{aligned} \min c^T x = z \quad & \text{subject to} \\ Ax \geq b, \quad & A \in \mathbb{R}^{m \times n} \\ x \geq 0 & \end{aligned} \tag{5.5}$$

is called a primal problem. Its dual problem is

$$\begin{aligned} \max b^T y = v \quad & \text{subject to} \\ A^T y \leq c, \quad & A \in \mathbb{R}^{m \times n} \\ y \geq 0 & \end{aligned} \tag{5.6}$$

The variables y of the dual problem (5.6) represent the constraints in the primal problem (5.5) and vice versa. The dual of the dual problem is the primal problem again. For the proof, simply transform the dual problem into a minimization problem and apply the above scheme to get the primal

problem again. The primal problem in standard form is stated as

$$\begin{aligned} \min c^T x = z \quad & \text{subject to} \\ Ax = b, \quad & A \in \mathbb{R}^{m \times n} \\ x \geq 0 & \end{aligned} \tag{5.7}$$

and its dual is

$$\begin{aligned} \max b^T y = v \quad & \text{subject to} \\ A^T y \leq c, \quad & A \in \mathbb{R}^{m \times n} \end{aligned} \tag{5.8}$$

For the proof, replace variable $y \in \mathbb{R}$ in the dual (5.8) by two non-negative variables: $y = y_1 - y_2, y_1 \geq 0, y_2 \geq 0$. This results in a dual problem in symmetric form like (5.8). Taking its dual gives problem (5.7). Several important theorems describe the close relation between feasible solutions to the primal and dual problems (for the proofs, see e.g. [30]).

Theorem 1. (Weak Duality Theorem)
If x^* is any feasible solution to the primal (5.5) and y^* is any feasible solution to the dual (5.6), then

$$y^{*T} b = v^* \leq z^* = c^T x^* \tag{5.9}$$

Proof. According to the definition of primal and dual

$$\begin{aligned} Ax^* &= b & \qquad y^{*T} A &\leq c^T \\ c^T x^* &= z^* & \qquad y^{*T} b &= v^* \end{aligned}$$

Multiplying $Ax^* = b$ by y^{*T} on the left and multiplying $y^{*T} A \leq c^T$ by x^* on the right gives

$$\begin{aligned} y^{*T} A x^* = y^{*T} b = v^* \\ y^{*T} A x^* \leq c^T x^* = z^* \end{aligned}$$

and thus

$$v^* = y^{*T} b = y^{*T} A x^* \leq c^T x^* = z^*.$$

□

In other words, the feasible solutions to the dual problems pose a lower bound for the solutions to the primal. The weak duality theorem implies the weak corollary that if $v^* = z^*$, then $v^* = \max v$ and $z^* = \min z$. The Strong Duality Theorem makes a much stronger claim.

Theorem 2. (Strong Duality Theorem)
If the primal system (5.7) has a feasible solution and the dual system (5.8) has a feasible solution, then there exist optimal feasible solutions x^* and y^* to the primal and dual systems such that

$$b^T y^* = \max v = \min z = c^T x^*. \tag{5.10}$$

Proof. Let matrix A of (5.7) be an $m \times n$ matrix with $\mathrm{rg}A = m$ and rows a_i. Let further be $N := \{1, \ldots, n\}$, and $J := \{j_1, \ldots, j_k\}$ an index vector of length k of pairwise different indices $j_i \in N$ for $1 \leq i \leq k$. Then, $A_J = [a_{j_1}, \ldots, a_{j_k}]$ is the $m \times k$ matrix of rows in A whose column indices are in J. Let K be complementary vector to J, i.e., $|J| + |K| = |N| = n$ and every $i \in N$ is either in J or in K. For very vector $x \in \mathbb{R}^n$, $x_J := (x_{j_1}, \ldots, x_{j_k})^T$. Thus, we have $Ax = A_J x_J + A_K x_K$. An index vector J is called basis of A if $|J| = m$ and A_J is a regular matrix. The variables x_{j_k}, $1 \leq k \leq m$ are called basic variables. The complementary index vector K is called non-basis and the corresponding variables are non-basic variables.

The equation $Ax = b$ is solvable and matrix A has at least one basis J and complementary non-basis K. It follows that

$$b = Ax = A_J x_J + A_K x_K \leftrightarrow A_J^{-1} b = A_J^{-1} Ax = x_J + A_J^{-1} A_K x_K$$

and thus the solution x of $Ax = b$ can be given with $\overline{b} := A_J^{-1} b$, $\overline{A} := A_J^{-1} A$, and $\overline{A}_J = A_J^{-1} A_J = I$ as

$$\overline{b} = x_J + \overline{A}_K x_K$$

The solution x in which $x_K := 0$ and $x_J := \overline{b}$ is called basis solution of $Ax = b$ and J. It is denoted $x(J)$. Basis J is a feasible basis and thus x a feasible solution if $x \geq 0$. By construction of x, equations $Ax = b$ are satisfied and $x_K \geq 0$. Thus, only $x_J \geq 0$ has to be checked. The pair $(J; [\overline{A} \;\; \overline{b}])$ is called the *simplex tableau* of basis J.

Problem (5.7) can be transformed into the so-called extended simplex form

$$\begin{gathered} \max z \quad \text{s.t.} \\ \begin{pmatrix} A & 0 \\ c^T & 1 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \\ x \geq 0 \end{gathered} \tag{5.11}$$

by introducing an auxiliary variable z. The Simplex algorithm calculates an optimal solution for problem (5.11), i.e., it finds an optimal basis vector J

and the corresponding basis solution $x(J)$ that maximizes $z = c^T x(J)$. The final simplex tableau for basis J is

$$\begin{pmatrix} \overline{A} & 0 & \overline{b} \\ \overline{c}^T & 1 & \beta \end{pmatrix} = \begin{pmatrix} A_J & 0 \\ c_J^T & 1 \end{pmatrix}^{-1} \begin{pmatrix} A & 0 & b \\ c^T & 1 & 0 \end{pmatrix}$$

With

$$\begin{pmatrix} A_J & 0 \\ c_J^T & 1 \end{pmatrix}^{-1} = \begin{pmatrix} A_J^{-1} & 0 \\ -c_J^T A_J^{-1} & 1 \end{pmatrix}$$

we obtain $\overline{b} = A_J^{-1} b = x_J$ and $\beta = c_J^T A_J^{-1} b = c_J^T x_J = z$. With $\pi := c_J^T A_J^{-1}$ the reduced costs can be written as $\overline{c}^T = -\pi A + c^T$. From the optimality of basis vector J and basic solution $x(J)$ and from the definition of the Simplex algorithm follows that the reduced costs of the basic and non-basic variables are

$$\overline{c}_J^T = -\pi A_J + c_J^T = 0 \tag{5.12}$$

$$\overline{c}_K^T = -\pi A_K + c_K^T \geq 0 \tag{5.13}$$

Hence, $\pi A \leq c^T$ and therefore π is a feasible solution to the dual (5.8). Because of the weak duality theorem, π is an optimal solution to (5.8). □

By transforming the dual problem (5.8) into primal form and reapplying the same proof, an optimal solution to the primal can be constructed from an optimal solution to the dual.

In the primal minimization problem (5.5), a constraint $A_i x = b_i$ is called *binding constraint* if incrementing b_i increases the optimal solution. When b_i is increased, the basic solution x_J remains feasible only within a certain interval around b_i. If e_i is the i-th unit vector and if $\epsilon \in \mathbb{R}$, the permitted values for $b_i + \epsilon e_i$ are given by the system of equalities

$$\begin{aligned} x'_J = A_J^{-1}(b + \epsilon e_i) &\geq 0 \\ \epsilon A_J^{-1} e_i &\geq -A_J^{-1} b \end{aligned}$$

Given an $\epsilon > 0$ that satisfies this set of equalities, changing the right-hand side b by ϵe_i does not affect the feasibility of the optimal dual solution y. However, changing the right-hand side may affect the objective value

$$c^T x' = y^T(b + \epsilon e_i) > y^T b \leftrightarrow y_i > 0.$$

Obviously, increasing the right-hand side of a constraint can only increase the objective value if the corresponding dual variable is positive. Does decreasing the right-hand side of a binding constraint also improve the objective value?

If problem (5.5) is non-degenerate, i.e., if both the primal and the dual solutions are unique, the positive dual variable y_i is both a necessary and a sufficient precondition for the improvement of the objective value if the i-th right-hand side is decreased. If the problem is degenerate, the positive dual variable is only a necessary but not a sufficient precondition.

Now, the lexicographic min-ordering problem (5.4) can be reformulated as a linear programming problem

$$\begin{aligned} &\max \tau \\ &\quad x_i \geq \tau, \forall i \in B' \\ &\quad x_j \geq t_j, \forall j \in B, t_j \in T \\ &\quad Ax \leq b \end{aligned} \tag{5.14}$$

which is equivalent to the problem

$$\begin{aligned} &\max (0\ 1) \begin{pmatrix} x \\ \tau \end{pmatrix} \\ &\quad \begin{pmatrix} -I_{B'} & 1 \\ -I_B & 0 \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \tau \end{pmatrix} \leq \begin{pmatrix} 0 \\ -T \\ b \end{pmatrix} \end{aligned} \tag{5.15}$$

and whose dual problem is

$$\begin{aligned} &\min (0\ -T\ b)y \\ &\quad \begin{pmatrix} -I_{B'} & -I_B & A \\ 1 & 0 & 0 \end{pmatrix} y = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &\quad y \geq 0 \end{aligned} \tag{5.16}$$

In each case, submatrices I_B and $I_{B'}$ are the partially filled identity matrices where diagonal element $a_{ii} = 1$ if and only if $i \in B$ or $i \in B'$ respectively.

Now, let vector y be partitioned into three parts $y := (y_{B'}\ y_B\ y_R)^T$ such that

$$\begin{pmatrix} -I_{B'} & -I_B & A \\ 1 & 0 & 0 \end{pmatrix} y = \begin{pmatrix} -I_{B'} \\ 1 \end{pmatrix} y_{B'} + \begin{pmatrix} -I_B \\ 0 \end{pmatrix} y_B + \begin{pmatrix} A \\ 0 \end{pmatrix} y_R$$

Thus, if $y_{j_k} > 0, j_k \in B'$ and $y_{j_k} \in y_{B'}$ then for constraint $x_{j_k} \geq \tau$ of problem (5.14), the equality $x_{j_k} = \tau$ holds. The value of τ cannot be further improved unless x_{j_k} is fixed at $t_{j_k} = \tau$ and j_k is added to indices B.

5.2.4 Solving the Scaled Layout Problem

Algorithm 21: Solve lexicographic scale maximization

Input: A set of scaled constraints S, with scale variables s_k, hard constraints H, and an upper bound u

Output: Set $S' \subseteq S$ containing only constraints that are redundant given $s_k \leq u$, set H extended with previously active constraints $S \backslash S'$

```
1  begin
2      I ← {k | e_k ∈ S}
3      while I ≠ ∅ do
           Solve   max   scale
                   scale  ≤ u
4                  scale  ≤ s_k   ∀k ∈ I
               subject to  S, H
5          if scale = u or scale is unbounded then
6              H ← H ∪ {s_k ≥ u | k ∈ I}
7              S' ← {e_k | e_k ∈ S ∧ k ∈ I}
8              return (H, S')
9          end
10         Let A = {i | y_i > 0, y is dual solution of above LP}
11         AC ← {e_i | e_i ∈ S ∧ i ∈ A}
12         z* ← objective value of above LP
13         I ← I \ A
14         S ← S \ AC
15         H ← H ∪ AC
16         H ← H ∪ {s_i ≥ z* | i ∈ A}
17     end
18     return (H, ∅)
19 end
```

Algorithm 21 shows how the linear lexicographic min-ordering problem is solved. Set I maintains the indices of remaining scale variables s_k whose value has not yet been fixed. The scale variable *scale* is maximized subject to the constraints in S, H and the upper bound u. If the linear problem of line 4 is unbounded or if *scale* attains the given upper bound u, the algorithm adds hard constraints to guarantee the achieved solution $s_k \geq u$ and returns in S' the remaining soft constraints. Otherwise, set A contains the indices

of positive dual variables, i.e., the indices of binding constraints, that are removed from I. AC contains the binding constraints themselves, that are removed from S. The binding constraints are added to H as hard constraints together with their achieved solution z^*. The algorithm terminates if all constraints in S have been binding and have thus been removed, or if enough constraints have been removed from S so that the linear problem becomes unbounded.

The complete solution algorithm that uses the Solve procedure defined in Alg. 21 is shown as Alg. 22. It summarizes the necessary steps to collect the scaled constraint set $\mathcal{SC} := \{SC_{\text{Min}}, SC_{\text{Max}}, SC_{\text{Equal}}, SC_{\text{Gap}}, SC_{\text{Close}}\}$. Each contained set of constraints is passed to the iterative solution algorithm together with the corresponding upper bound on the scale variables. With each call to the solution algorithm, the set of hard constraints H is extended. Each call to Solve is passed the scale variable upper bound as an argument. The fixed constraints are then added to H. With each call to Solve, the feasible region is restricted further.

The solution algorithm is called twice for the gap constraints. With the first call, a very small upper bound of, e.g., 1/64th of the page extent, is passed to the solve algorithm, i.e., a minimum gap between adjacent gridlines that should be guaranteed. In the first run, all gaps that cannot attain this upper bound are fixed. The remaining soft gap constraints are returned in set S'_{Gap}. Then, the outer-most gridlines are moved as close as possible to the page boundaries considering the previously guaranteed minimum gap. Finally, all remaining space is filled by the remaining gap constraints with no upper bound.

5.3 Underspecification

As noted before, inserting gap constraints only between those gridlines whose occupied spans intersect leads to under-constrained solutions in very simple problem instances as the one shown in Fig. 5.5. In this example, two shapes are inserted diagonal to each other so that neither in horizontal nor in vertical direction their occupied spans intersect. No gap constraints are inserted between the shapes and the optimization problem stated above would give a solution that places both on top of each other.

The insertion of any gap constraint would imply an a priori order on the gridlines which may be unintended. Instead, after each run of the layout solver, the ICBM layout system checks for collisions between gridlines. The first solver run on the input of Fig. 5.5 will place both rectangles on top of each other. The collision detection is essentially a variant of the line segment

Algorithm 22: Solve scaled layout problem

Input: The user defined constraints $\mathcal{C} = \{C_{\text{Min}}, C_{\text{Max}}, C_{\text{Equal}}, C_{\text{Merge}}\}$ and page P

begin
  // Collect gap constraints, see Section 5.2
  $\mathcal{C} \leftarrow \mathcal{C} \cup \text{AddGapConstraints}(\mathcal{G}_x, P.w_x.v - P.0_x.v)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \text{AddGapConstraints}(\mathcal{G}_y, P.h_y.v - P.0_y.v)$
  // Introduce scale variables, see Section 5.2.1
  $\mathcal{SC} := \{SC_{\text{Min}}, SC_{\text{Max}}, SC_{\text{Equal}}, SC_{\text{Gap}}, SC_{\text{Close}}\}$
  $H \leftarrow \emptyset$
  $H \leftarrow \text{Solve}(SC_{\text{Min}}.S, H \cup SC_{\text{Min}}.H, SC_{\text{Min}}.u).H$
  $H \leftarrow \text{Solve}(SC_{\text{Max}}.S, H \cup SC_{\text{Min}}.H, SC_{\text{Max}}.u).H$
  $H \leftarrow \text{Solve}(SC_{\text{Equal}}.S, H \cup SC_{\text{Equal}}.H, SC_{\text{Equal}}.u).H$
  foreach *partition* $(v, f, G) \in C_{Merge}$ **do**
    foreach *gridline* $g_i \in G$ **do**
      $H \leftarrow H \cup \{g_i.v = v\}$
      if *H infeasible constraint set* **then**
        $H \leftarrow H \setminus \{g_i.v = v\}$
      end
    end
    $G' \leftarrow \{g_i \mid g_i \in G \wedge g_i.v = v\}$
    $g_M \leftarrow g \in G'$
    $g_M.f \leftarrow f$
    foreach *gridline* $g_i \in G'$ **do**
      Merge g_i into g_M
    end
  end
  $(H, S'_{\text{Gap}}) \leftarrow \text{Solve}(SC_{\text{Gap}}.S, H \cup SC_{\text{Gap}}.H, \epsilon)$
  $H \leftarrow \text{Solve}(SC_{\text{Close}}.S, H \cup SC_{\text{Close}}.H, SC_{\text{Close}}.u).H$
  $\text{Solve}(S'_{\text{Gap}}, H, \infty)$
end

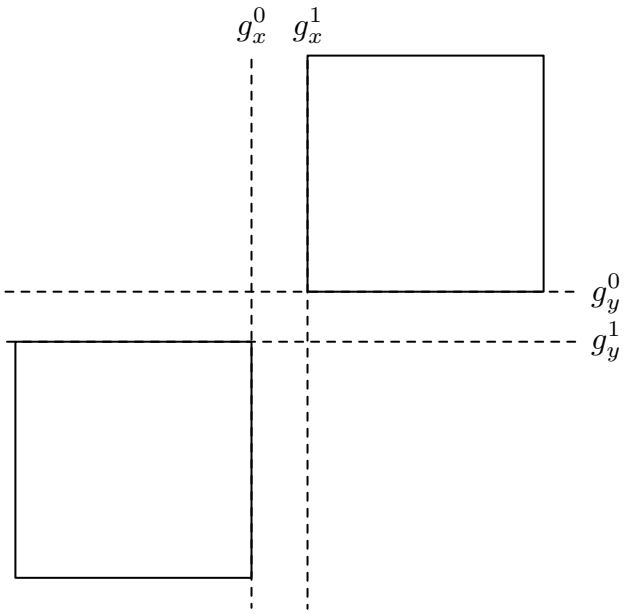


Figure 5.5: No gap constraints are inserted between the gridlines and both shapes will be placed on top of each other by the layout solver.

intersection algorithm [12]. It sweeps over the set of all gridlines and finds, e.g., that gridlines g_1 and g_2 have a different order now than they had in the input. Additionally, both gridlines have occupying spans that intersect. This shows that the positions of gridlines g_1 and g_2 have been under-constrained. Because the occupying spans overlap, both gridlines should have collided. If v^1_{pre} and v^2_{pre} are the original positions of gridlines g_1 and g_2, and v^1_{post}, v^2_{post} are the corresponding positions after the solver run, the gridlines collide when

$$\lambda v^1_{\text{post}} + (1-\lambda)v^1_{\text{pre}} = \lambda v^2_{\text{post}} + (1-\lambda)v^2_{\text{pre}}, 0 \leq \lambda \leq 1.$$

The algorithm searches the smallest such λ over all collisions. If the movement between position v^i_{pre} and v^i_{post} is understood as a continuous movement, λ is the time of the first collision between any two gridlines and all gridlines are moved back to the time of the earliest collision. Another iteration of the solution algorithm will add gap constraints based on the changed gridline positions, resolving the previous ambiguity. This algorithm proceeds until the gridline positions do not change anymore. The algorithm terminates because the gridlines are moved to the interpolated values at which at least two of them collide. That collision causes the insertion of a new gap constraint which defines the order of two gridlines in every subsequent solution.

Algorithm 23: Checking for collisions between gridlines

Input: Gridlines $\mathcal{G}_x, \mathcal{G}_y$

begin
  while *gridline positions not stabilized* **do**
    solve layout problem
    $G_x \leftarrow \mathcal{G}_x,\ G_y \leftarrow \mathcal{G}_y$
    // let $v_{pre}(g_i)$ be the original value for gridline g_i and $v_{post}(g_i)$ the value calculated by the layout solver
    sort G_x, G_y such that $g_i < g_j$ if and only if $v_{\text{pre}}(g_i) < v_{\text{pre}}(g_j) \vee (v_{\text{pre}}(g_i) = v_{\text{pre}}(g_j) \wedge v_{\text{post}}(g_i) < v_{\text{post}}(g_j))$
    $T \leftarrow 0$
    $T_x \leftarrow \emptyset,\ T_y \leftarrow \emptyset$
    foreach *pair of successive gridlines* $g_i, g_{i+1} \in G_{x/y}$ **do**
      $t_i \leftarrow \infty$
      if $v_{pre}(g_i) < v_{pre}(g_{i+1}) \wedge v_{post}(g_i) \geq v_{post}(g_{i+1})$ **then**
        // there was a collision
        $\triangle v_{\text{pre}} \leftarrow v_{\text{pre}}(g_i) - v_{\text{pre}}(g_{i+1})$
        $\triangle v_{\text{post}} \leftarrow v_{\text{post}}(g_i) - v_{\text{post}}(g_{i+1})$
        $t_i \leftarrow \triangle v_{\text{pre}} / (\triangle v_{\text{pre}} - \triangle v_{\text{post}})$
      end
      $T_{x/y} \leftarrow T_{x/y} \cup \{t_i\}$
    end
    while *true* **do**
      find $g_i \in G_x \cup G_y$ with the smallest $t_i \geq T$
      $T \leftarrow t_i$
      if $T > 1$ *or occupied spans of* g_i, g_{i+1} *intersect* **then break**
      swap g_i and g_{i+1}, $t_i \leftarrow \infty$, update t_{i-1} and t_{i+1}
    end
    move all gridlines to interpolated values at time T
  end
end

5.4 Summary

This chapter presented the main contribution of this thesis: a general-purpose page layout algorithm that solves the layout problem defined on page 44. Given an instance of the layout problem that may include explicit user-defined constraints, the layout algorithm first approximates the permitted widths and heights of every text-containing shape using a set of linear equations. The step function of narrowest width-height-configurations of a text box are computed using Dynamic Programming. The layout algorithm analyzes the topology of the user-drawn shapes in the layout problem instance and adds gap constraints that restrict shapes from moving through each other. The layout problem is then solved by exploiting the analogy to resource allocation problems. All constraints are converted into so-called soft constraints by introducing a negative error term, called *scale* variable. The scale variable of a minimum distance constraint can be interpreted as the degree of satisfaction. If the scale variable is less than one, the constraint is not satisfied. In gap constraints, the scale variable corresponds to the size of the gap between two shapes. The layout algorithm tries to maximize the scale variables, i.e., to maximize the constraint satisfaction and the size of the gaps. Since the available page space is limited, not all distance constraints may be satisfiable and gaps cannot be made arbitrarily large. The layout algorithm follows a lexicographic min-ordering strategy to achieve an equitable distribution of constraint errors and gap sizes. The lexicographic min-ordering optimization problem can be solved by solving a sequence of simpler optimization problems. All constraints are given as linear equations and thus, every subproblem is a linear optimization problem. The sequence of linear optimization problems can be generated and solved efficiently because the binding constraints in each subproblem can be identified quickly by analyzing the solution to the dual linear optimization problem.

By transforming hard constraints into soft constraints, the layout algorithm can handle over-constrained problems. Infeasible constraints cannot be satisfied completely but the remaining constraint error is minimized and distributed among related constraints. The layout algorithm handles under-constrained problems by analyzing the calculated layout. The layout algorithm avoids the difficulties of determining a priori which parts of a layout problem might be under-constrained and instead searches for shapes in the finished layout that have moved through each other, i.e., shapes that collided. The layout algorithm adds the minimum number of constraints to disambiguate the problem until a stable and conflict-free solution is found.

The layout algorithm is capable of solving real-world layout problems with several thousand constraints in real-time.

Chapter 6

Evaluation

The ICBM layout system is able to handle a wide variety of document layouts. All sample layouts in the following section are based on real-world examples. The samples illustrate the capabilities and limitations of the current ICBM layout algorithms.

6.1 Qualitative Evaluation

6.1.1 Example A

The sequence of Figures 6.1-6.4 depicts the creation of the organizational chart shown in Fig. 6.4 with all intermediate steps:

1. The user selects a rectangle from the shape menu and clicks on the slide. The text box is centered automatically. The user adds the desired text and changes the background color. The text color is adapted automatically, as shown in the first image of Fig. 6.1, to achieve a high contrast.

2. The user clicks on the rectangle to select it, presses the Ctrl key, and drags the rectangle downwards until the top of the dragged outline aligns with the bottom of the existing rectangle. The user releases the mouse button and the selected rectangle is copied. The copy is aligned with the previously inserted shape at the top, left, and right. The user changes the text of the copied rectangle and both shapes resize automatically as the second image of Fig. 6.1 shows.

3. The user selects both shapes, presses Ctrl, and drags downwards. The generated copy shown in the bottom image of Fig. 6.1 is aligned at the left and right to the original.

4. Again, the user selects both shapes and presses Ctrl-D for *duplicate*. An insertion indication appears and the user inserts both shapes at the bottom of the page, without any alignment. The two shapes are duplicated another time and inserted inside the last inserted text box as shown in top image of Fig. 6.2.

5. The internal shape is duplicated, or Ctrl-dragged, another four times to create the center image of Fig. 6.2.

6. The user selects the two internal shapes in the upper row and adds an equal width constraint.

7. This is repeated for the three internal shapes in the lower column.

8. The pair of heading and text box is duplicated again and aligned as shown at the top of Fig. 6.3.

9. The same pair is duplicated to form the two columns of internal shapes in the following image. Both columns are constrained to have the same width.

10. The user draws three connecting lines shown in the last image of Fig. 6.3.

11. The finished diagram of Fig. 6.4 contains a few more explanatory text boxes whose positions are only determined by their content and alignment constraints.

The result is shown in Fig. 6.4. Apart from the headline, which has been placed manually in the top left corner, no object in this sample has been manually resized or positioned. As described, the only user constraints are shape alignment and equal width constraints. All shape sizes and shape positions have been calculated automatically and the resulting layout looks very balanced. The lower image of Fig. 6.4 shows the resulting layout after a some text has been added in the lower left text boxes. The layout automatically adapts without any user intervention. Appendix A shows an example of a complete constraint system generated to create a layout as shown in Fig. 6.4.

Figure 6.5 shows the original real-world example that has been created by a human expert. The original content has been removed. Directly below is the layout as calculated by the ICBM layout algorithm. Without prior knowledge, it is impossible to tell them apart.

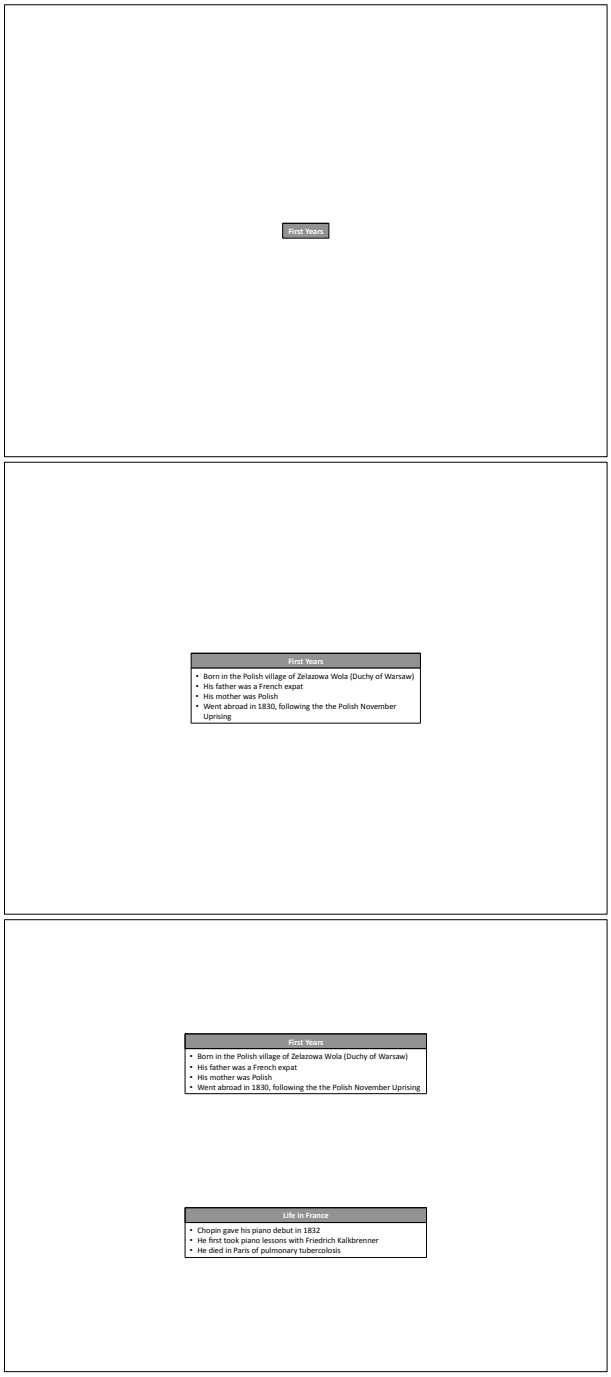


Figure 6.1: The first two text boxes are created as copies of each other and both are left- and right-aligned.

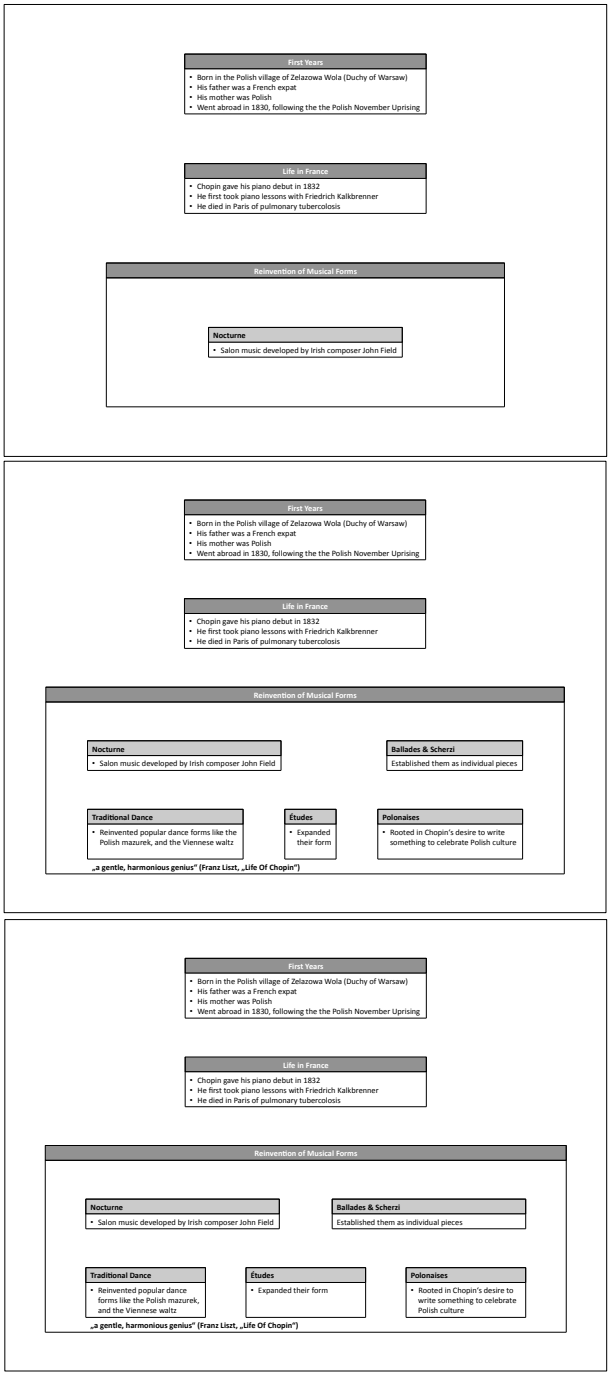


Figure 6.2: An existing text box is copied twice, one copy containing the other. The inner copy is then further duplicated. The inner text boxes are constrained to have equal widths.

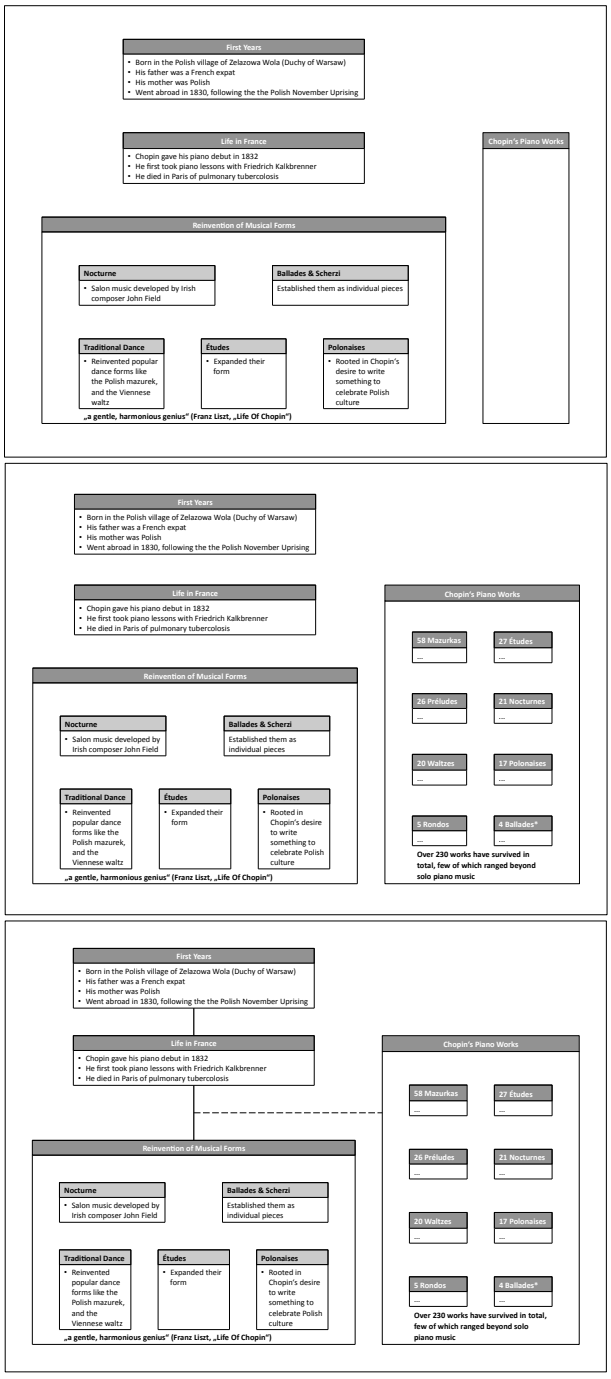


Figure 6.3: An existing text box is copied again and upon insertion it is aligned at the top and bottom to existing text boxes. Is is also filled with multiple copies of the same text box.

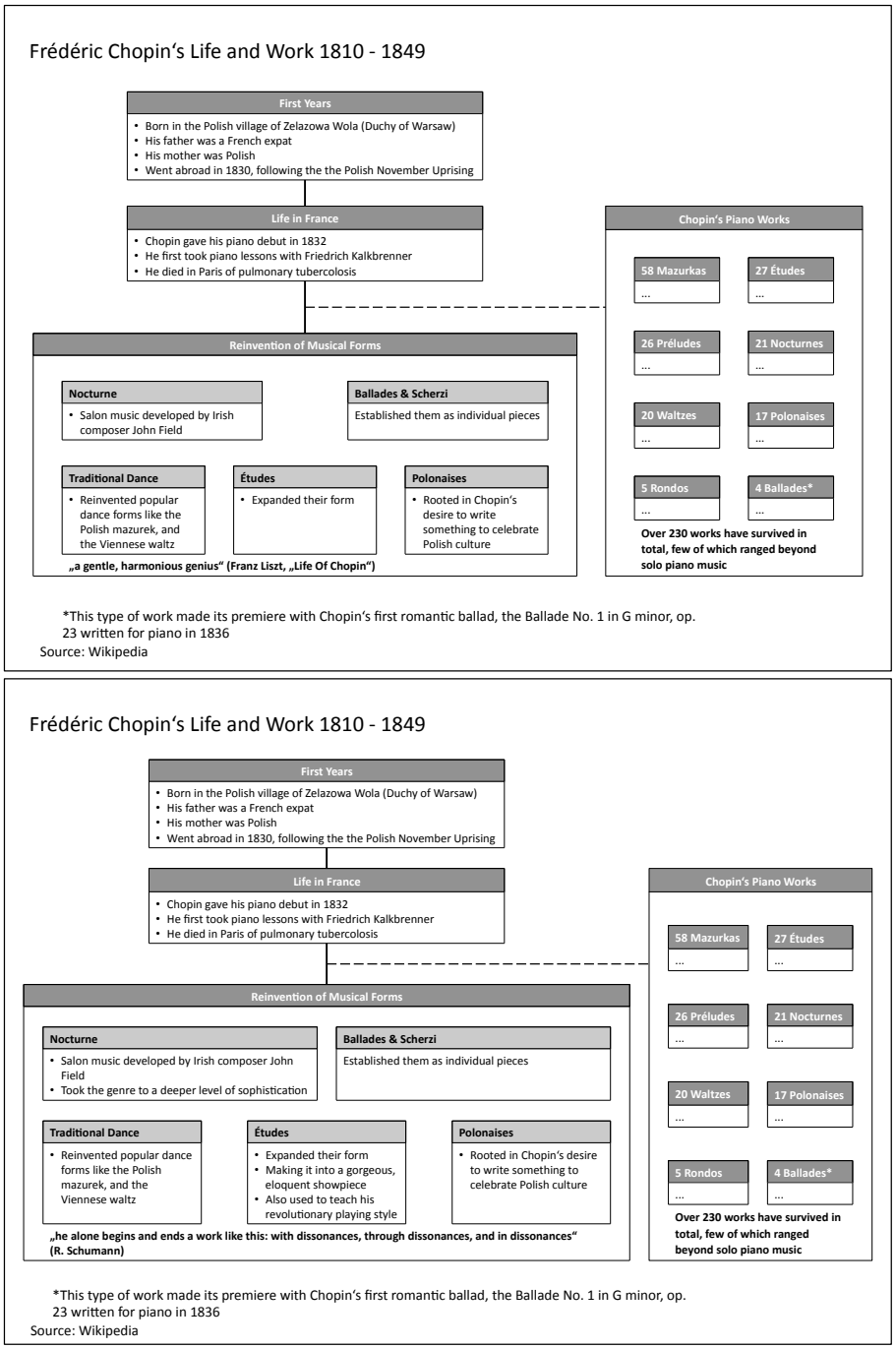


Figure 6.4: The final layout is seen above. If some of the text content changes as seen below, the layout adapts automatically.

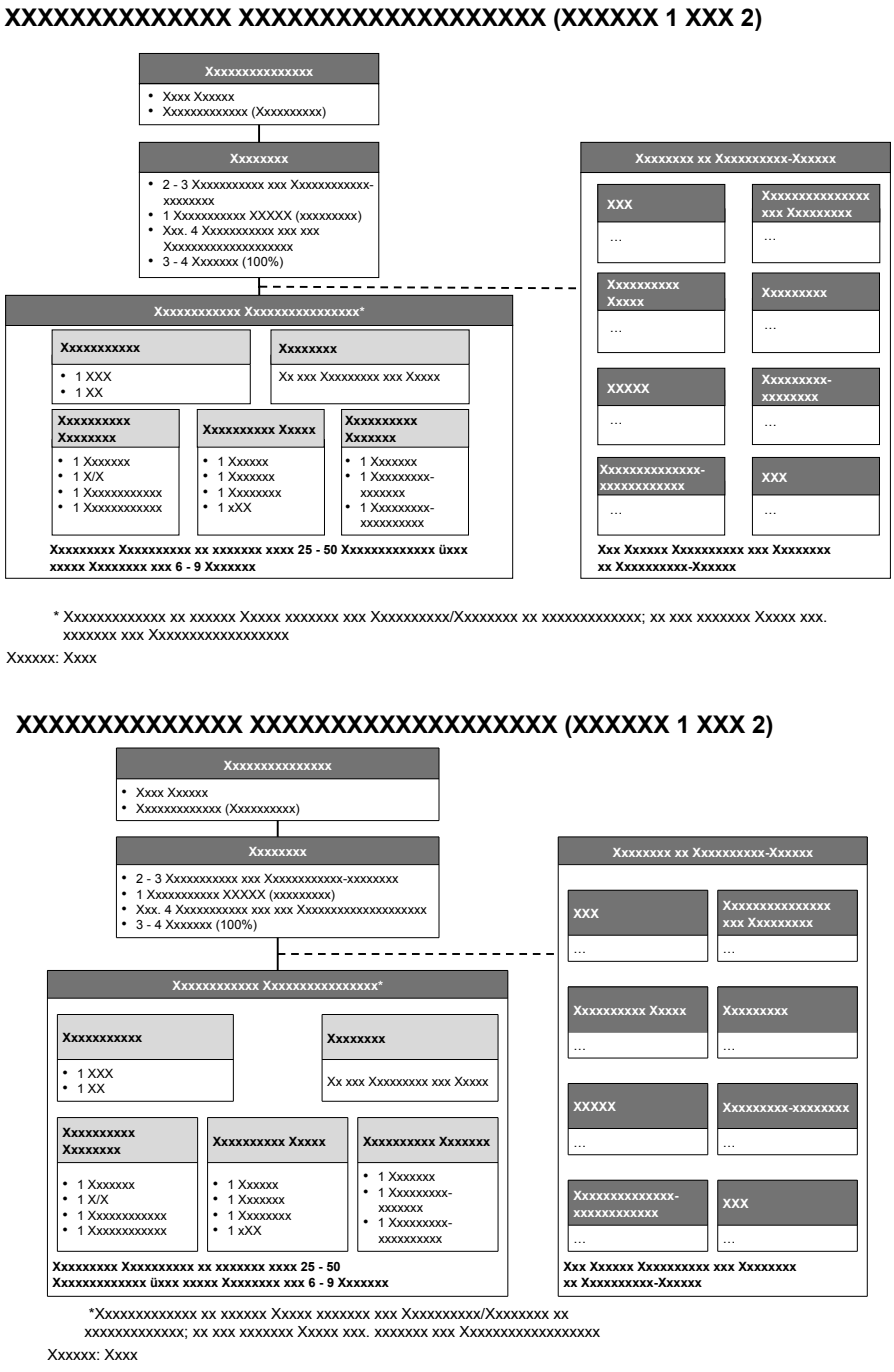


Figure 6.5: The same layout created by a human expert (above) and the ICBM layout algorithm (below).

6.1.2 Example B

The layout of Fig. 6.6 has been achieved without explicitly setting the position or size of any element except the headline. The top-most three objects (Johann Sebastian Bach and his two wives Maria Barbara and Anna Magdalena) are constrained to the same width and height. The text boxes forming the row below, containing five of the 10 children, are aligned at the top and bottom and are set to have equal width. The boxes representing the grandchildren are aligned at the right to their parent and they are aligned to each other at the top. The text boxes representing the remaining five children are arranged in two columns and three rows. Inside each column the boxes are aligned at the left and right, in each row the top and bottom coordinate are aligned. Furthermore, both columns are constrained to have equal widths. Again, Fig. 6.8 shows the original example as created by a human expert designer and the ICBM layout system.

6.1.3 Example C

The definition of a shape given on page 41 permits many different kinds of complex shapes. To create the example shown in Fig. 6.7, I defined an extended text box shape consisting of an inner and outer rectangle. The inner rectangle, defined by four inner gridlines, surrounds the text. The outer rectangle is also defined by four outer gridlines. The space between the inner and outer rectangle is the text margin. With this change, shapes can be inserted between the inner and outer margin as annotations to the text like the rounded rectangles of the example slide. The size of the text margin can be optimized automatically depending on the font size of the contained text.

6.1.4 Example D

Another problem that is elegantly solved by introducing interior gridlines that surround the text is shown in Fig. 6.9. In many cases, objects are meant to align directly with the text and not with the outside border of the shape. This example layout also shows that the ICBM system is not restricted to rectangular shapes. Pentagons have internal size constraints that always maintain the length of the tip in relation to the height of the shape.

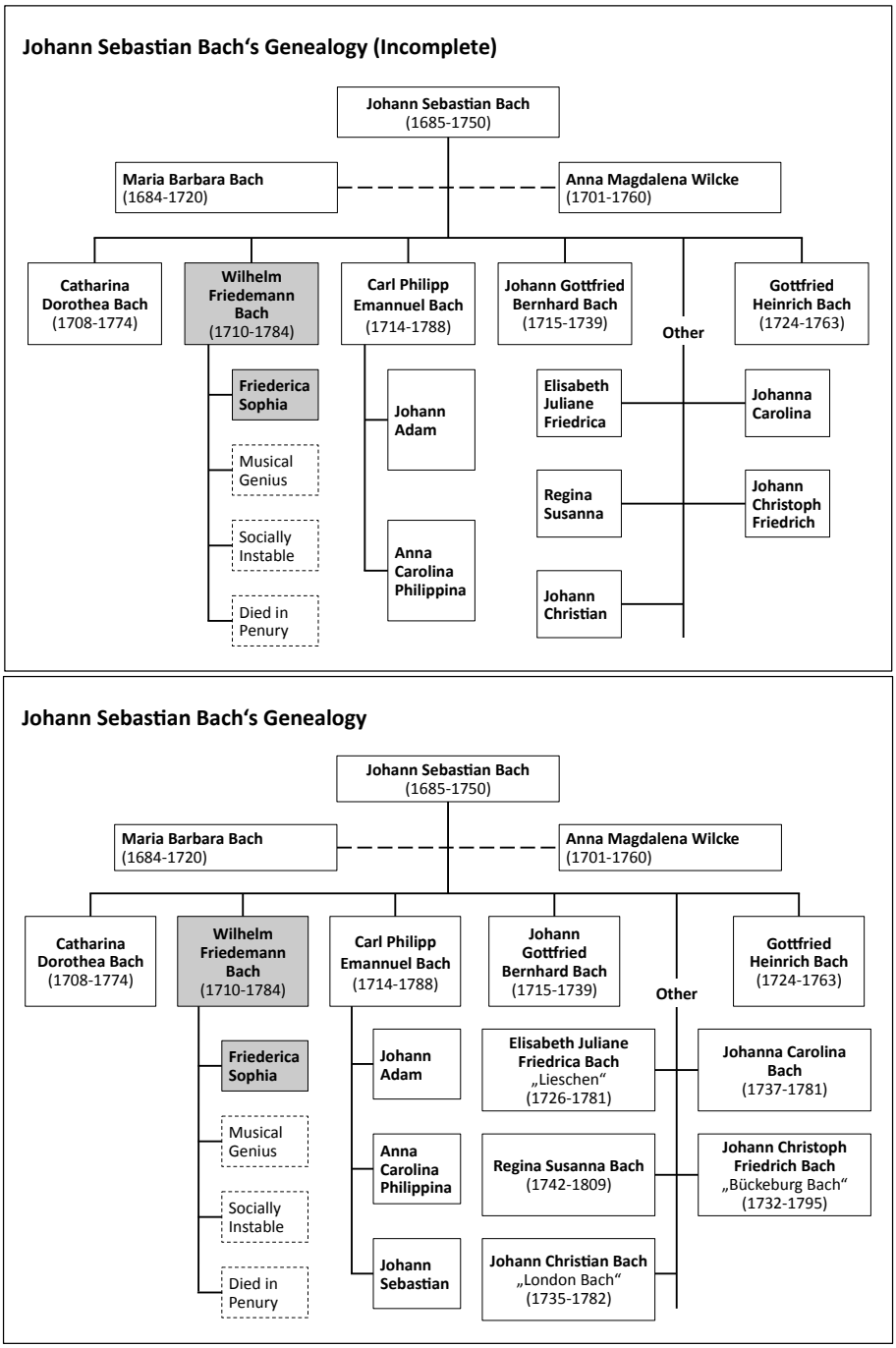


Figure 6.6: An organization diagram created with the ICBM layout system. The bottom image shows how the layout adapts to content changes.

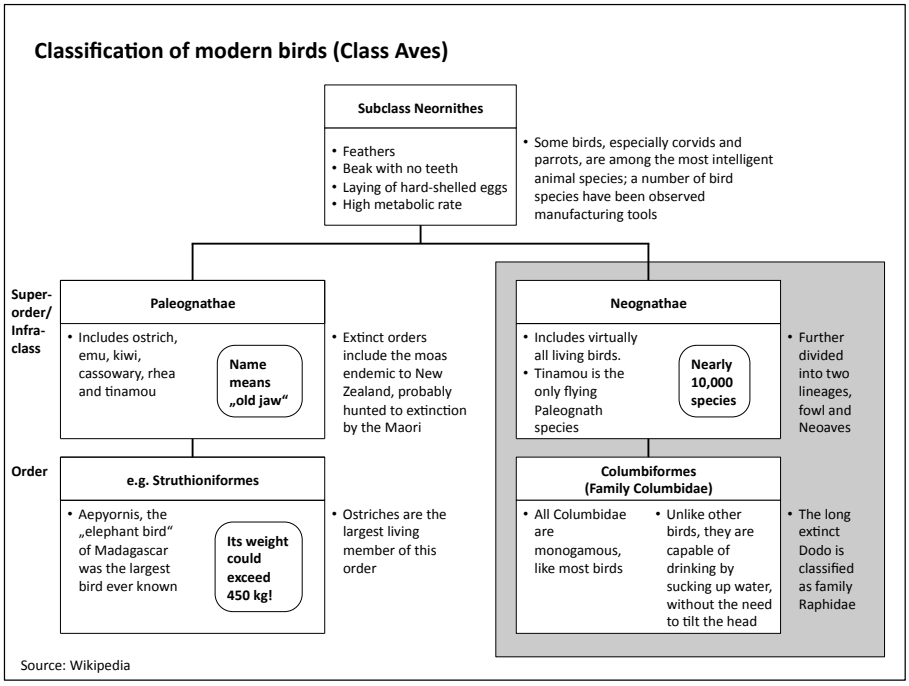


Figure 6.7: The rounded rectangles need to be positioned inside another text box without overlapping the text content.

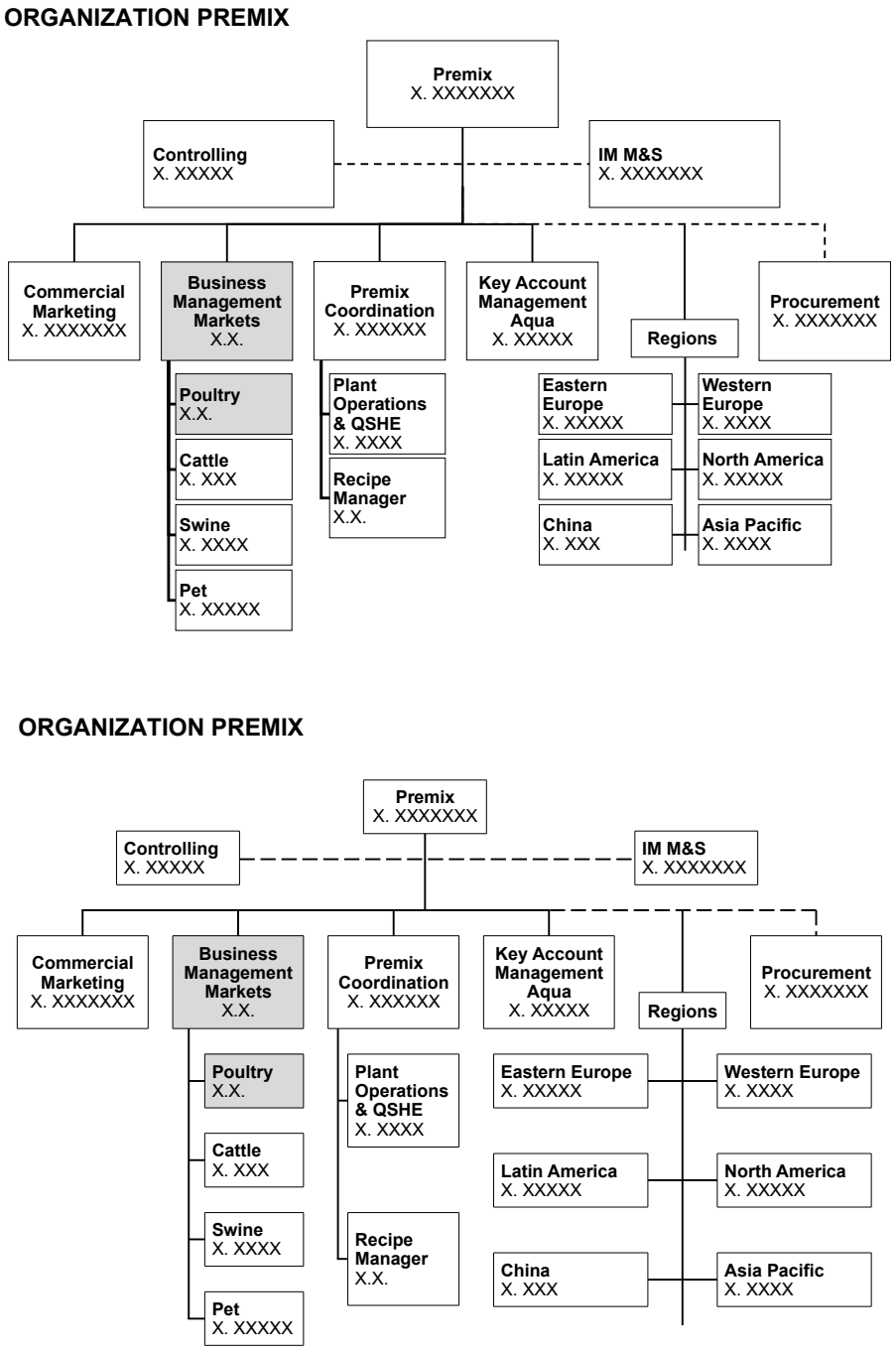


Figure 6.8: The same layout created by a human expert (above) and the ICBM layout algorithm (below).

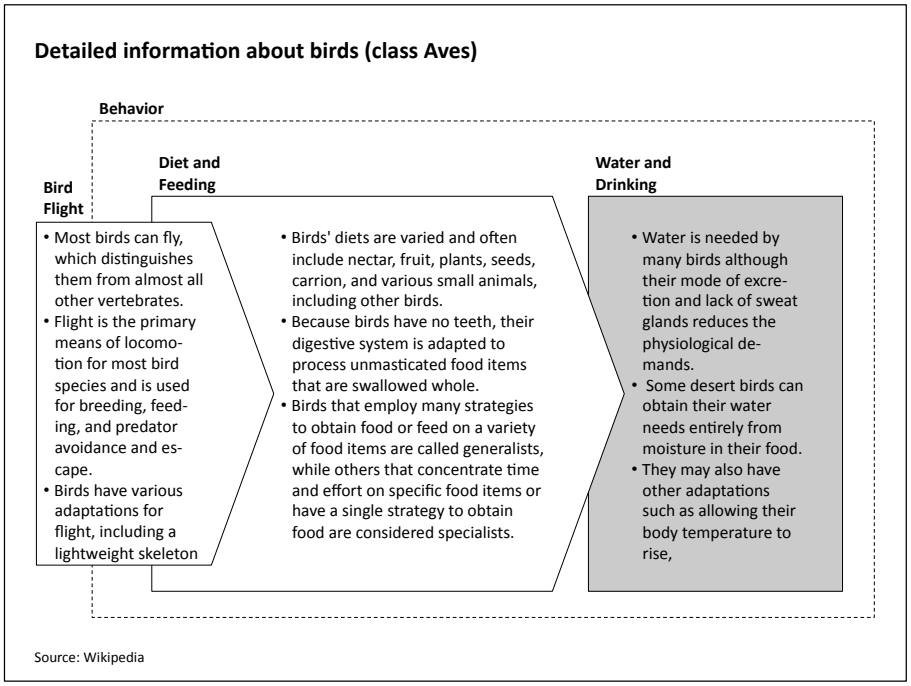


Figure 6.9: Pentagons can be aligned to the text content instead of the shape outline.

6.1.5 Example E

The final sample of Fig. 6.10 contains a complex arrangement of nested and aligned objects. Text boxes with invisible outlines are aligned to lines that connect text boxes and the whole ensemble is contained in large pentagons. The small multiplication sign is a simple drawing on top of the laid out shapes. It is not automatically positioned or sized. Shapes that benefit from the automatic layout functionality and shapes that do not can coexist in the same document. Standard shapes can serve as additional decorations.

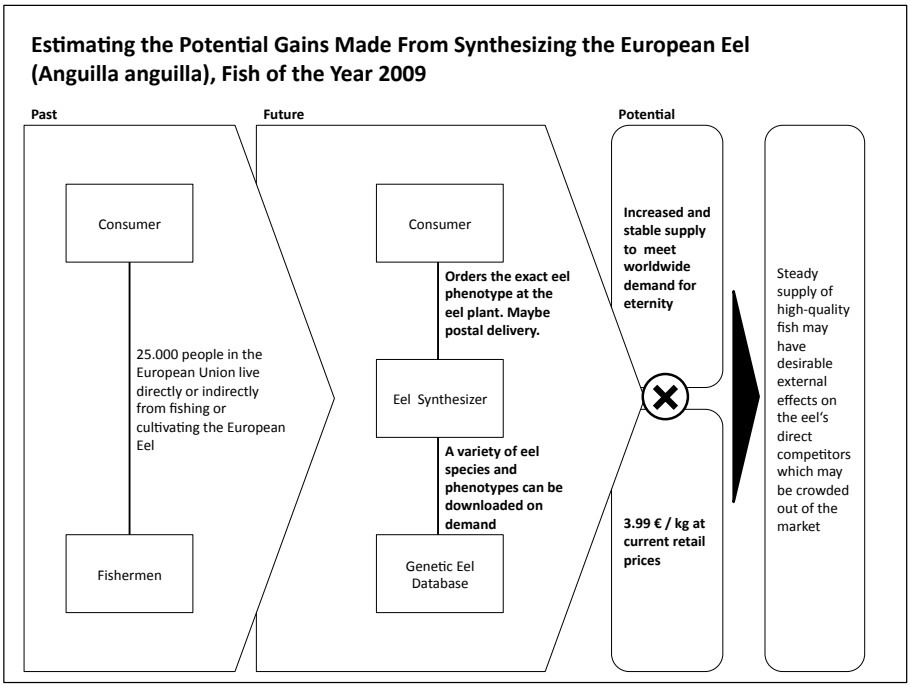


Figure 6.10: A combination of multiple design elements.

6.2 Quantitative Evaluation

The qualitative evaluation showed that the layout algorithm presented in this thesis is capable of solving complex real-world layout problems. Due to the diversity of the solvable layout problems, it is difficult to find a good quality metric for a quantitative evaluation. However, the layout algorithm can be evaluated on a well-researched subproblem with a clear quality metric that is relevant in the application domain of ICBM itself. The calculation

of table layouts with minimum height is a such a problem as the number of papers proposing both theoretical results and solution algorithms suggests [115, 3, 71, 69, 67]. It is also a relevant subproblem because the ICBM system is expected to handle the simple case of tabular layouts well. An evaluation experiment that compares the general ICBM system to specialized table layout algorithms can estimate how much the specialized algorithms gain from their additional application knowledge.

Definition 30. A **table cell** c is a list of words $W := \{x_0, \ldots, x_m\}$. w_i, h_i are defined to be the width and height of word x_i. The minimum width of c is the width of the widest word $x_i \in W$: $\text{minwidth}(c) := \max_{x_i \in W}(w_i)$. The minimum height is similarly defined to be the height of the highest word.

Definition 31. A **table** is a tuple $T := (C, \text{rows}, \text{cols})$ where C is the set of cells according to Def. 30 the table T is composed of. $\text{rows} : C \to \mathbb{N}$ is the function that returns for each cell in $c \in C$ how many rows c spans. Correspondingly, $\text{cols}(c)$ is the number of columns spanned by c.

Definition 32. Given a table T, the **Minimum Height Table Layout Problem** is the problem of assigning each cell $c_i \in T.C$ a width wc_i and height hc_i that are large enough to fit the list of words $x_j \in c_i.W$, such that the total height of T is minimal.

As noted before on p. 26, the list of words in each cell only permit a discrete number of minimal width-height-configurations. Finding the minimum-height table layout thus requires solving a combinatorial optimization problem that is NP-complete. All practical table layout algorithms therefore approximate the permitted table cell sizes.

6.2.1 Tight Table Layouts

In a recent publication, Hurst et al. presented four different table layout algorithms [71]. The area approximation algorithm (AA) exploits the fact that the table cell area remains approximately constant over all minimal width-height configurations. The constant text area constraint can be expressed using a convex cone constraints. Linear constraints are used to guarantee each cell's minimum width and height. The conjunction of linear constraints and cone constraints is also convex. Commercial solvers like Mosek [95] are capable of solving systems of conic and linear constraints.

The iterative column widening algorithm (ICW) begins by setting each table cell to its minimum width. Then, the algorithm iteratively reduces the table height by calculating the next widest table configuration that reduces

the table height by at least one line. Both algorithms use completely independent strategies that are combined by the authors to create several hybrid algorithms.

The hybrid of area-approximation and iterative column widening (AA-ICW) uses the area approximation algorithm to compute an initial table layout and then tries to improve the result by using ICW. The hybrid HTML-ICW algorithm used a simple table layout strategy to compute the initial table layout. The HTML table layout algorithm (HTML) calculates a table layout by resizing each column proportionally to the number of contained text.

The four algorithms and the simple HTML algorithm were evaluated by the authors on a set of seven sample tables. Each table had to be rendered for a range of maximum allowed table widths in the range of 450 - 1200 pt. The authors compared both the achieved table height and the performance of all five algorithms to the table layout engine of the Mozilla web browser. The Mozilla table layout engine performed much worse than all algorithms as Fig. 6.11 shows.

The ICBM layout algorithm differs in several aspects from table layout algorithms. First of all, given a constraint system representing a table, the ICBM system does not explicitly try to minimize the table height. While this optimization criterion is useful for tables, it is not really applicable to general document layouts. The ICBM system will try to fit the table into the surrounding space, i.e., the whole page. If there is a lot of free space, the ICBM system may insert gaps inside some table cells, thus resizing them to occupy more space than strictly necessary. If the available space is very limited, the user would expect the ICBM system to find a table configuration with minimum width and height.

Second, both the table layout algorithms and the ICBM system will layout a table with the minimum width if the desired width is less than the minimum required width. If, however, the desired width is very large, the table layout algorithms will never exceed the maximum necessary width, i.e., the width achieved if every paragraph in every cell occupied only a single line. The ICBM system will always fulfill the width constraint if it is feasible. If necessary, the table cells will receive a very large margin.

Third, if the iterative column widening algorithm has found the smallest table height for a given table width, it will in turn also calculate the smallest necessary table width to achieve the same height. Thus, the desired width is interpreted as a maximum width and the resulting table width will typically be less than that. These differences must be taken into account while designing the evaluation experiment.

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(Mozilla)

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X) HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(ICBM)

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X) HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(AA-ICW)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(Mozilla)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(ICBM)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(AA-ICW)

Figure 6.11: Even on simple tables, Mozilla performs worse than the ICBM layout algorithm. The AA-ICW algorithm achieves tables that are both narrower and flatter.

6.2.2 Experimental Setup

Nathan Hurst, Kim Marriott and Peter Moulder were so kind to provide their original implementations of all five table layout algorithms together with the set of sample tables they had designed to evaluate the different algorithms. In their evaluation, each table was rendered with its maximum width set to values between 450 pt and 1200 pt in steps of 50 pt. The achieved heights were compared relative to the table height achieved by the Mozilla HTML renderer. Since all algorithms presented by Hurst et al. performed much better than the Mozilla renderer, it is no longer included in this evaluation.

Originally, the table layout code read a sample table from an XHTML file, calculated the optimal table layout, and created a PDF file as output. The table layout code was integrated into the ICBM system itself, because both have to use the same font measurement to achieve comparable results.

An automated test run reads all sample XHTML files using the original code, calculates the table layout with each of the five algorithms and for each desired width and outputs each result in a format that can be read by the ICBM layout algorithm. The documents are then opened again and the table layout is recomputed by the ICBM layout algorithm.

6.2.3 Results

In the first evaluation experiment, each sample table is rendered by each algorithm with the desired table width differing in the range of 450 pt to 1200 pt. The results achieved by each algorithm for each sample at 800 pt width are shown in Appendix B. Table 6.1 shows the achieved table height in points for a table of 800 pt width. Table 6.2 shows the achieved heights over all widths relative to the results obtained by the ICBM algorithm.

Example	ICBM	HTML	AA	ICW	HTML-ICW	AA-ICW
2n2-linear	58	58	58	58	58	58
multipara	144	201	158	144	144	144
simple-brick	58	58	58	58	58	58
cs-schedule	144	144	144	129	129	129
counterfeit	1093	1179	1078	1078	1078	1078
diagonal5	158	173	158	129	129	129
columns	4946	1121	992	992	1049	992
plants200	4370	4773	4327	4140	4327	4255

Table 6.1: Table heights in points for the maximum table width set to 800 pt.

Example	ICBM	HTML	AA	ICW	HTML-ICW	AA-ICW
2n2-linear	100%	100%	98%	97%	97%	97%
multipara	100%	134%	100%	97%	97%	97%
simple-brick	100%	108%	100%	100%	100%	100%
cs-schedule	100%	104%	96%	91%	91%	91%
counterfeit	100%	106%	99%	97%	97%	97%
diagonal5	100%	103%	100%	87%	87%	87%
columns	100%	24%	22%	21%	23%	21%
plants200	100%	106%	102%	98%	102%	101%

Table 6.2: Table heights over all maximum widths relative to the result if the ICBM layout algorithm.

The chosen samples differ in size between the small *2n2-linear* sample that requires only four lines of text at 800 pt width and the huge *plants200* sample, a table of 200 rows. The simple HTML algorithm performs worse than the ICBM system on all of the sample tables, the *columns* sample being an exception that is analyzed below. On the *multipara* sample, the HTML algorithm calculated table layouts that were 34% higher than those rendered by the ICBM system.

The area approximation algorithm calculated table layouts of similar height compared to the ICBM system, not taking into account the *columns* example. The area-approximation algorithm and the ICBM system show identical results on three out of seven samples. On another three samples, the area approximation algorithm had given results that were 1%, 2% and 4% better. In the largest sample, the table calculated with the area approximation algorithm was 2% higher that the table calculated by the ICBM algorithm.

All three algorithms that use the iterative-column widening algorithm either alone or in a hybrid approach consistently outperform the ICBM layout algorithm. In the case of the *diagonal5* sample, the iterative-column widening algorithms calculated tables of 13% less height on average. This synthetic sample shown in Fig. 6.12 has been constructed to show how close an algorithm's results are to the known optimal solution.

A Look at the Results for the *column* Sample

The *column* sample is shown in Fig. 6.13. The left table has been calculated by the area approximation algorithm, and the right table has been calculated by the ICBM system. The unnecessarily large table calculated by the ICBM

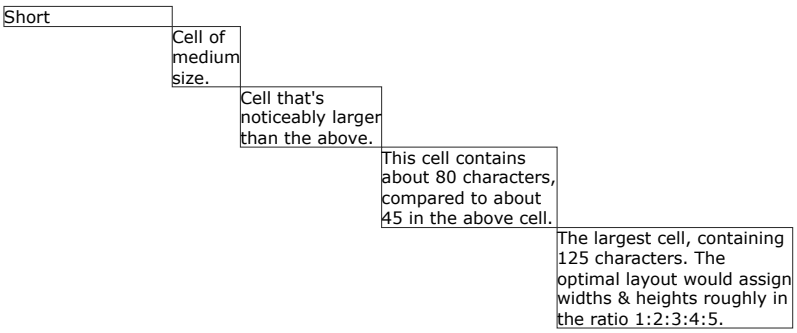


Figure 6.12: Constructed sample *diagonal5* tests how close a layout algorithm's results are to the optimal solution.

layout algorithm is an artifact of the experimental setup. In order to fit the 200 rows *plants200* sample, the page size for every table rendered by the ICBM layout algorithm was set to over 2000 pt. The top-left corner of each table was aligned with the top-left corner of the page. With the right table edge constrained to the desired table width, only the bottom edge of each table could be moved by the ICBM algorithm. The maximization of the gap between the bottom edge of the page and the bottom edge of the table effectively minimized the table height.

In the *column* sample, the gap constraint insertion algorithm of page 99 added two gap constraints inside the table: between the bottom of the first cells in the first and second column, and the second and third column respectively. These additional gaps were optimized together with the gap below the table. Because a lot of page space was available, the layout algorithm stretches all gaps equally to fill the available page space. This poses two questions: First, if the page were smaller, how compact could the ICBM layout algorithm make the table? And second, why are the gap constraints inserted?

How small can the ICBM system make the table? A binary search on the table height resulted in a minimum height of 1006 pt for a table width of 800 pt. This is very close to the results obtained by the area approximation algorithm, the iterative column widening algorithm and the hybrid of both, the AA-ICW algorithm. Each of these three algorithms achieved a table height of 992 pt. The result is better than that achieved by the HTML algorithm alone and in its hybrid variant. On average over all samples and widths, the minimum height obtainable by the ICBM system is just 2% larger than the height achieved by the hybrid AA-ICW algorithm. Thus, if the page layout is very dense, the ICBM system calculates very compact tables.

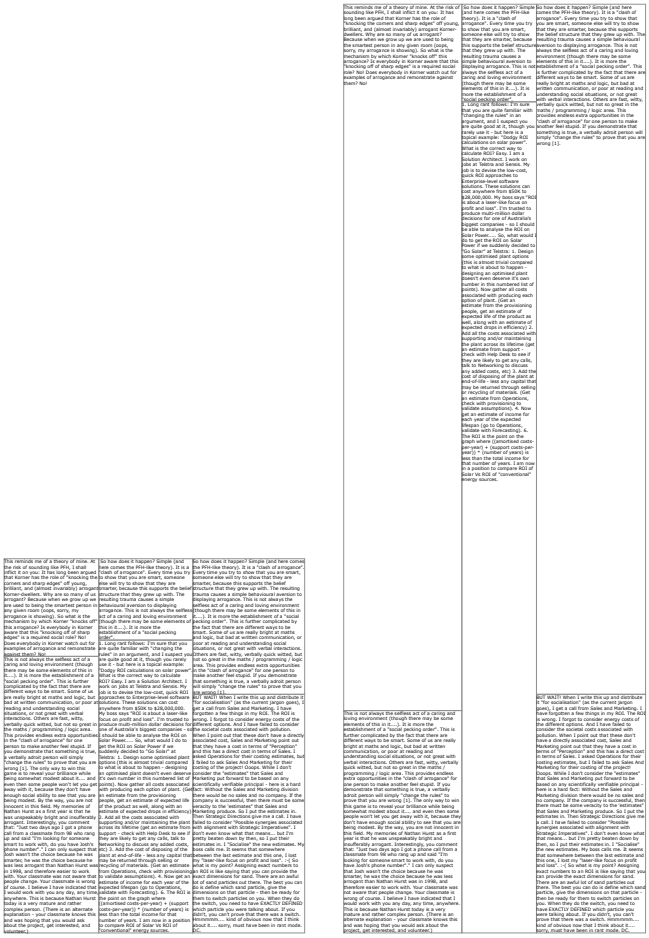


Figure 6.13: Example *columns* rendered by the area approximation algorithm (left) and the ICBM system (right).

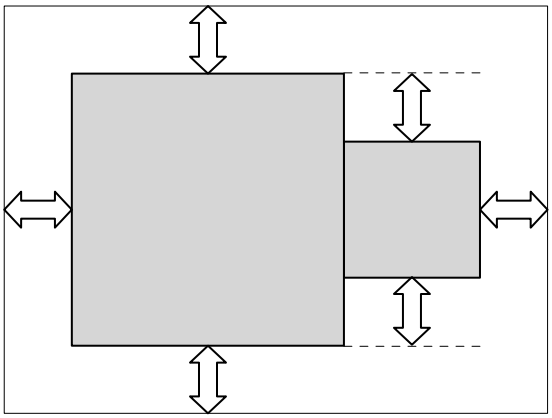


Figure 6.14: The two rectangles show the gap constraints that lead to the solution of example *columns*.

In its current implementation, the gap insertion algorithm constrains the positions of two adjacent shapes that touch each other. In Figure 6.14, the depicted gaps restrict the possible movement of the right rectangle. Its top and bottom cannot move past the top and bottom of the left rectangle due to the gap constraints. While this decision is sensible for the general case, it constrains the order of table cells unnecessarily.

Minimizing the Table Width

The results so far showed that the iterative column widening algorithms performed better than the ICBM layout system. As previously mentioned, the iterative column widening algorithms calculate a minimum height configuration for a table of *at most* the given width. Once a minimum height configuration is found, the iterative column widening algorithms also calculate the minimum width necessary to achieve the same height. The iterative column widening algorithms usually calculate tables of strictly less than the desired width. Thus, the results shown in Table 6.3 underestimated the capabilities of the iterative-column widening algorithm.

In a second experiment, the ICBM system had to layout each sample with the table width set to the narrowest width obtained by the iterative column widening algorithm. The results are shown in Tables 6.3 and 6.4.

The results show that the iterative column widening algorithms perform better than the ICBM system, calculating tables up to 21% lower than the tables obtained by the ICBM algorithm. In practice, the iterative column widening algorithms often obtained tables that in total required a single line

Example	ICBM	ICW	HTML-ICW	AA-ICW
2n2-linear	72	58	58	58
multipara	158	144	144	144
simple-brick	72	58	58	58
cs-schedule	144	129	129	129
counterfeit	1121	1078	1078	1078
diagonal5	158	129	129	129
columns	4946	992	1049	992
plants200	4370	4140	4327	4255

Table 6.3: Table heights in points with the desired table width set to the narrowest table width found by ICW.

Example	ICBM	ICW	HTML-ICW	AA-ICW
2n2-linear	100%	79%	79%	79%
multipara	100%	91%	91%	91%
simple-brick	100%	83%	83%	83%
cs-schedule	100%	87%	87%	87%
counterfeit	100%	96%	96%	96%
diagonal5	100%	83%	82%	82%
columns	100%	21%	23%	21%
plants200	100%	98%	101%	101%

Table 6.4: Table heights relative to the result of the ICBM layout algorithm with the desired table width was set to the narrowest table width found by ICW.

less. In a sample with only four lines of text, such as the *2n2-linear* and *simple-brick*, calculating a table layout requiring only three lines would constitute a 25% improvement. In the larger sample tables such as *counterfeit* and *plants200* the advantage of using the iterative-column widening algorithms is relatively smaller.

Analysis

Why do the iterative-column algorithms outperform the ICBM layout algorithm? Could the performance of the ICBM layout system be improved? Consider the example of table *2n2-linear* shown in Fig. 6.15. In a first phase, the ICBM layout algorithm approximates the discrete width-height-configuration of each table cell. The text in the first column requires the column to have widths and heights of at least (190 pt, 90 pt) or (210 pt, 80 pt). The second column is required to be at least (270 pt, 90 pt) or (290 pt, 80 pt). Because the possible sizes are only approximated using linear functions, the algorithm could assign the first column a width of 200 pt and a height of 103 pt. The second column is 300 pt wide and 103 pt high.

The second phase of the layout algorithm finds the exact text size closest to approximate solution. Thus, the first column is constrained to be at least 190 pt wide and 90 pt high. Correspondingly, the second column is constrained to be 290 pt wide and only 80 pt high. This result is not optimal because the total width of 500 pt is sufficient to make both columns only 80 pt high.

The ICBM algorithm cannot find this solution because it cannot trade the width of one cell for the reduced height of another cell. The algorithm is unaware that both cells form a table, i.e., that both objects belong to a single logical object and that their sizes should be optimized together. The iterative column widening algorithm widens one column to decrease the total table height. The ICBM solver can try to find the most compact text size for an individual cell but currently each decision is local and tradeoffs between different text boxes are impossible.

Performance

The performance measurements for each algorithm are shown in Table 6.5. The times include building the constraint set and solving the layout problem but they do not include rendering the output. The numbers show little differences between the algorithms. The conic programming approach is slightly slower than the ICBM layout algorithm, the HTML algorithm is noticeably faster in both variants but it is within the same order of magnitude, and the

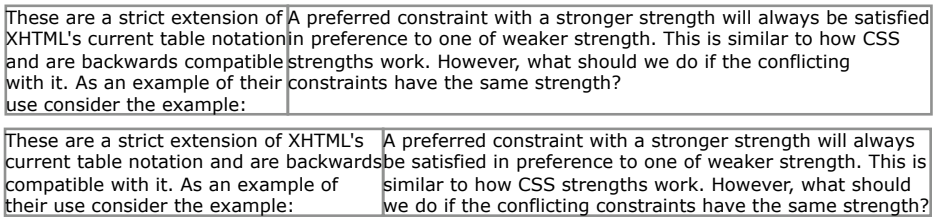


Figure 6.15: Example *2n2-linear* is rendered with one more line by the ICBM algorithm (top) compared with the iterative column widening algorithm (bottom).

iterative column widening algorithms are on par with the ICBM system. The performance of all algorithms is dependent on the size of the input. Solving the *2n2-linear* sample is much faster than solving the *plants200* example. The numbers do not reveal however, that the results are completely dominated by the time it takes to measure the text extents. Because the ICBM implementation is integrated into Microsoft PowerPoint, the text measurements needs to be done in PowerPoint.

Example	ICBM	HTML	AA	ICW	HTML-ICW	AA-ICW
2n2-linear	29	32	44	31	30	43
multipara	70	85	99	88	85	100
simple-brick	46	39	51	40	39	52
cs-schedule	68	73	89	75	73	89
counterfeit	211	248	272	254	250	274
diagonal5	30	29	43	30	29	44
columns	559	643	661	720	649	664
plants200	2219	1153	1257	2414	1292	1333

Table 6.5: Time in milliseconds taken by each algorithm to render each sample table for the 15 different widths between 450 pt and 1200 pt.

Table 6.6 shows the calculation times again, this time excluding the time to measure the text extents. As expected, the trivial HTML implementation gives almost instantaneous results. The hybrid HTML variant and the iterative column widening algorithm are very fast for small examples but with larger samples, containing more text, the performance of the iterative column widening algorithm deteriorates severely. That is consistent with the results reported by the authors themselves.

Example	ICBM	HTML	AA	ICW	HTML-ICW	AA-ICW
2n2-linear	6	0	12	0	0	12
multipara	10	0	12	2	0	12
simple-brick	9	0	12	0	0	12
cs-schedule	16	0	14	1	0	14
counterfeit	51	0	21	5	2	22
diagonal5	10	0	13	1	0	13
columns	63	0	13	67	4	16
plants200	1434	1	104	1252	143	180

Table 6.6: Time in milliseconds taken by each algorithm to render each sample table excluding the time necessary for text measurements.

On small samples, the ICBM system is slightly faster than the area approximation algorithm which is probably due to the performance penalty of using an interior point algorithm supporting conic constraints. On the *columns* and *plants200* samples, the AA algorithm outperforms the ICBM system and the iterative column widening algorithm by a significant margin.

The area approximation algorithm requires very few constraints because there is only one area constraint per table cell compared with a large number of approximate linear constraints. The results could indicate that conic constraints are superior both in terms of performance and in terms of quality to the approximation of text sizes using linear constraints. There are several reasons why we think linear solvers are the better choice.

First, Table 6.6 shows the time required by a single conic solver run. As chapter 5 describes, solving the lexicographic maximization problem requires at least five solver runs, one for each set of constraints, which completely annihilates the seeming performance advantage of conic constraints. Much of the performance overhead of the ICBM layout algorithm is necessary, because the ICBM layout algorithm tries to solve a much more complicated problem than the table layout competitors. Second, for practical reasons, the availability of many high-quality linear programming solvers is a big advantage. If one solver shows numerical instabilities or if a solver turns out to be too slow, it can easily be exchanged for another solver with essentially the same capabilities. Third, conic area constraints cannot be transformed into soft constraints and a layout system using conic constraints for approximating text sizes would therefore have difficulties in dealing with over-constrained constraint systems.

The quantitative analysis shows that the ICBM layout algorithm can calculate table layouts of comparable and sometimes even superior quality

compared with special purpose table layout algorithms. With a two-phase optimization, the linear approximation of text cell sizes yields effectively the same results as an area approximation algorithm. Using a linear approximation algorithm instead of conic area constraints allows the use of a huge range of fast linear solvers. The iterative column widening algorithm can find more compact table layouts by exploiting knowledge about tables that is not accessible to the ICBM system. The competing table layout algorithms are faster, because they do not have to handle infeasible constraint systems, non-compact document layouts, or underspecified solutions as does the ICBM system

6.3 Limitations of the Layout Algorithm

The evaluation showed that the proposed layout algorithm can still be improved in several ways. In the ICBM system, each shape's constraints are completely independent of its neighbors. If multiple shapes together form a table, it could be preferable to generate a slightly different constraint set, but at the moment, the ICBM system does not recognize such a high-level structure. Because every shape is attached to gridlines that are shared among aligned shapes, it would be possible to analyze the structure of shapes and gridlines to identify a table.

The problem became visible in very simple tables such as Fig. 6.15 shown on page 142. As the text accompanying the illustration explained in more detail, the layout algorithm first calculates an approximate width for each table column that is rounded to the closest realizable minimum-size text configuration. This rounding is performed independently for each text cell and the layout algorithm cannot increase one cell's width to achieve an overall preferable table layout.

Running the ICBM layout algorithm on tables also produced an unsatisfactory result for the *columns* sample table shown in Fig. 6.14, page 139. ICBM has inserted vertical gap constraints between the table cells as illustrated in Fig. 6.16.

Because of the gap constraints, the sample table's height is inflated to fill the available page space. In the previous chapter, I have argued that if two shapes are attached to the same gridline in one dimension, their edges in the other dimension have to be separated by gap constraints. Several cases can be distinguished. Either both shapes are separated vertically, as in Fig. 6.17a, they are adjacent (b), overlap partially (c), completely (d), or one is contained in the other (e) as in gap constraints between the edges of aligned shapes. In cases (b) and (d), both shapes are aligned vertically and

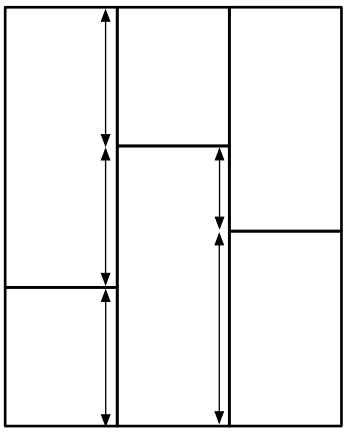


Figure 6.16: Gap constraints between overlapping table cells.

no gap constraints are inserted. In the other three cases, the gap constraints achieve a regular distribution of space and in case (e), the gap constraints keep the smaller shape centered inside the larger shape.

The gap constraint algorithm could, after a careful analysis of the problem, be improved to detect, e.g., the unnecessary gap constraints between interior shape edges. A sequence of vertically aligned text boxes as shown in Fig. 6.18 could be treated as a single object such that gap constraints are only inserted between the extremal edges of these combined objects, not between each individual text box as before. Thus, depending on the amount of text in each table cell, both results shown in Fig. 6.18 were realizable.

Another problem that became apparent during the evaluation is the overly regular distribution of gap space. In Figure 6.6 on page 127, several text boxes are made larger than necessary, especially the boxes labeled "Johann Adam" and "Anna Carolina Philippina" are too large. Each box is attached to a connector line. For each text box, two gap constraints between the connector and the top and bottom edge center the connector line vertically. Centering the connector is clearly desirable, but by inserting a gap constraint, the space between the connector line and the boxes edges is maximized together with all other gaps on the page.

Visually, the gap around the connector line and the gaps between text boxes don't have the same importance. The layout algorithm could be improved by defining several gap subsets that are maximized independently. Both gaps around a single connector line could form their own gap set, centering the connector line. Another class of gaps could be introduced to model text margins with a preferred size relative to the text size.

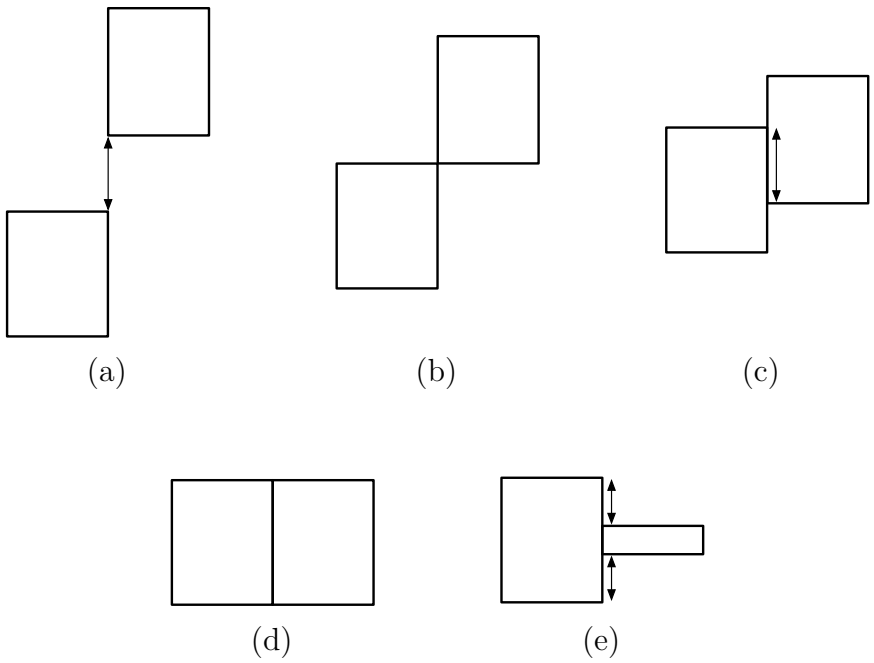


Figure 6.17: Two table cells can be vertically arranged in five different ways.

6.4 Summary

The layout examples A - E at the beginning of this chapter show the variety of layout problems that the lexicographic min-ordering layout algorithm solves. Obviously, the capabilities of the ICBM layout algorithm far exceed the capabilities of simple table layout algorithms. All five samples are based on real-world slides from an international consulting company. The calculated layouts are indistinguishable from the original layouts created by human experts. The finished layouts remain flexible and adapt gracefully to changed content or even changed page sizes. Previous layout systems such as the ALM system [87, 86] could express layout problems of similar complexity, but they neither provided the algorithms to create and manipulate problem instances nor the sophisticated layout algorithms necessary to solve such layout problems.

All samples have been created using the ICBM system of course. Through its intuitive interface the user can quickly explore many different layouts by rearranging the page elements. The layout system frees the user from the tedious task of adapting the size and position of every element on the page after each modification. The user is free to concentrate on the page content

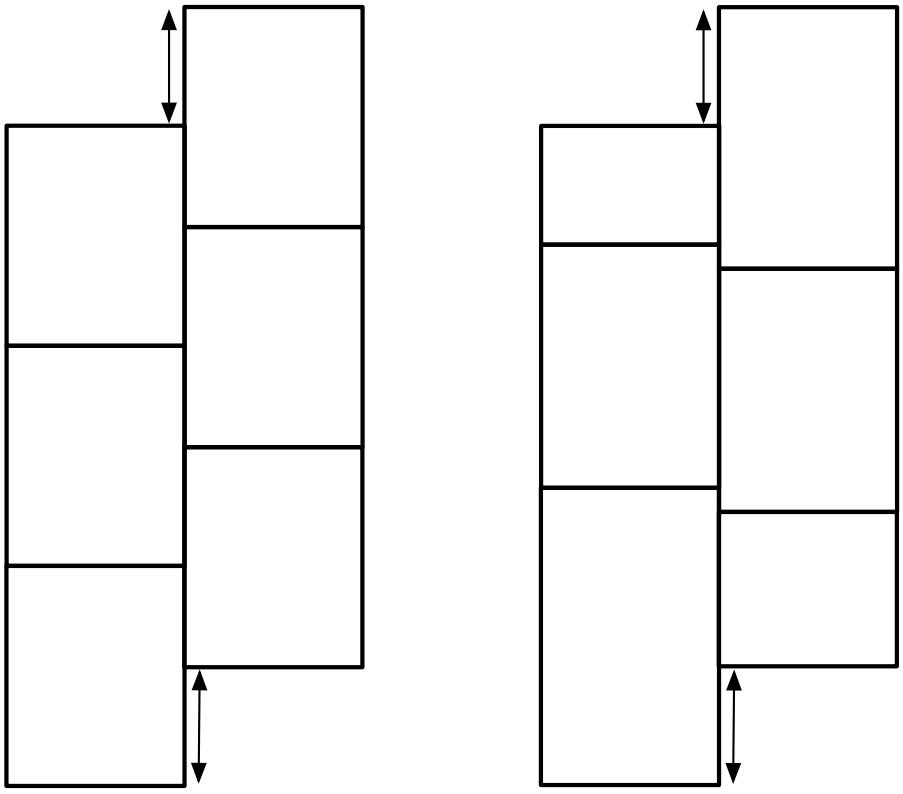


Figure 6.18: Desired gap constraints between table cells that do not fix the order of inner gridlines.

and the logical structure, the ICBM system is responsible for maintaining the desired layout.

The quantitative evaluation has shown that despite its superior capabilities, the ICBM layout algorithm does not perform significantly worse than optimized table layout problems. The evaluation showed that, if necessary, the presented layout algorithm is capable of calculating very compact table layouts. In general, the ICBM layout algorithm has been designed for the more complex task of optimizing the table layout together with the layout of the whole page. The width and height of the table are therefore dependent on the amount of available page space, on the page dimensions, on alignments of the table to other visual elements on the page etc.

Chapter 7

Conclusion and Future Work

> *"Whatever diminishes constraint diminishes strength. The more constraints one imposes, the more one frees one's self of the chains that shackle the spirit."*
> Igor Stravinsky

The ICBM system frees the user from having to specify the minutiae of slide layouts. It combines the user's creativity with the computer's attention to detail. The user can quickly explore different layouts, choosing the one that best fits a combination of the content's meaning, the user's personal style, and the audience's expectations.

The layout algorithm calculates the precise positions and sizes, the user describes only spatial relations and supplies the content. The user can move objects, realign and redistribute them, rewrite their content, delete some of them or add more and the layout algorithm adjusts every object on the slide instantaneously, making sure every object has the space it needs.

The user can manipulate shapes with familiar interaction techniques. Objects can be selected, dragged, rotated, or resized. When shapes are cut, copied, and pasted, the constraint relations between the selected objects are maintained. Shapes can be squeezed into arbitrarily small space, even between two previously aligned shapes, and the layout algorithm will find a new distribution of space to accommodate the additional object. Conversely, when shapes are deleted, the remaining empty space can be collapsed, aligning adjacent objects if they were only separated by the removed object.

This thesis presented a general, geometric class of layout problems that is capable of expressing the previously distinct classes of layout problems that arise in the creation of presentation slides, i.e., table layout problems, flow chart layout problems, and organizational chart layout problems. An instance of the general layout problem is a rough sketch of a presentation slide

consisting of shapes and manually specified constraints over these shapes. Chapter 4 showed the elementary operations required to create and manipulate layout problem instances programmatically and interactively. The chapter contained algorithms for inserting, moving, and manipulating sets of shapes together with the constraints between them. These algorithms have been used to implement the direct manipulation ICBM user interface. The interface avoids many of the pitfalls of earlier constraint-based drawing applications by clearly separating the responsibilities of the user and the layout system: The user sketches a slide layout and the layout algorithm maintains the topology of the user's drawing, but neither the exact positions nor sizes of the user's sketch are maintained.

The main contribution of this thesis is the layout algorithm presented in chapter 5. First, the user's sketch is analyzed by the gap constraint insertion algorithm. Gap constraints are inserted between neighboring shapes. Their extent is automatically calculated by the layout algorithm. The permitted sizes of all shapes containing text are approximated by linear constraints using an efficient Dynamic Programming algorithm. The user-specified constraints, text approximation constraints, and gap constraints are transformed into soft constraints by introducing scale variables that act as negative error terms. The layout problem is thus transformed into a resource allocation problem where the available page space is the primary resource being distributed among the constraints. The scale variables are maximized using a lexicographic min-ordering optimization strategy that can be solved in real-time when all constraints are given as linear equations. If the layout problem has been over-constrained, this optimization strategy finds a solution that distributes the infeasibility error evenly among many constraints. The solution layout is analyzed for errors caused by an under-constrained subproblem and the minimum number of constraints is inserted to transform the under-constrained problem into an unambiguous layout problem.

The resulting layouts are flexible and can adapt to changing content, font sizes, or even to different page sizes without further user intervention. The evaluation in chapter 6 shows that the presented layout algorithm is capable of solving a diverse set of layout problems with results indistinguishable from those achieved by human experts.

7.1 Further Applications

Obviously, the presented algorithms can be used to construct tools that solve only subproblems of the general layout problem tackled in this thesis. The evaluation chapter has shown that the presented layout algorithm can cal-

culate table layouts as compact as the solutions achieved by optimized table layout algorithms. A table layout tool based on this thesis would automatically offer an interactive table editor that lets the user draw complex tables with cells spanning multiple columns or rows. Table cells would not be limited to simple rectangles but could be arrows, or even charts.

The definition of the general layout problem is sufficiently general to fit completely different application domains, such as the user interface layout problem. The common user interface toolkit elements such as labels, buttons, check boxes, radio boxes, and windows can be modeled using gridlines and few simple constraints. Labels and buttons obviously resemble text boxes, even if a button may have rounded corners. Check boxes and radio boxes are fixed size squares.

The presented layout algorithm is already used in practice by 20.000 users of *think-cell chart* to layout so-called agenda slides. When giving a presentation, it is common practice to begin with a slide showing the topics that the presentation will cover. This agenda slide is repeated at the beginning of each new topic to give the audience a sense of orientation. Usually, all agenda slides contain the same text, but the current topic is highlighted. In some cases, sub-topics of all but the current topic are hidden. It is very cumbersome to edit agenda slides manually as any change to the text has to be repeated on all slides. Although each agenda may have different dimensions, due to different displayed sub-topics and different formatting, all agendas must be left- and top-aligned. On all slides, the identical topic text must break at the same position although its formatting, and thus width, may be different. The think-cell agenda tool is based on the work done for this thesis and it proved to be a very challenging problem for the layout algorithm. Because all agenda slide layouts have to be calculated together, the layout constraint sets become extremely large.

7.2 Future Work

Work on the ICBM layout system continues and it is focussed on two main aspects. First, the user interface will be further improved thanks to the feedback from early user tests. This work will hopefully lead to a formal usability study. In such a complex system, there is a delicate balance between the usability of the interface and the capabilities of the layout algorithm. When the layout algorithm does not produce the results a user desired, she will try to achieve manually what the system could not do automatically. If this proves unexpectedly difficult, the user will complain about the user interface, and probably not about the layout algorithm. Second, the layout algorithm

will of course be improved continuously. The user's problem specification contains much more information than the algorithm analyzes at present and the layout algorithm could make better use of existing rules of good layout that define the legible length of a line of text, or the optimal margin around text of a certain font size.

The interactive slide layout system described in this thesis forms the basis for the commercial *think-cell layout* application. A first version of think-cell layout will hopefully be released to users before the end of 2010. Several international consulting companies have shown strong interest after testing early prototypes.

Bibliography

[1] Alpern, B. ; Hoover, R. ; Rosen, B. K. ; Sweeney, P. F. ; Zadeck, F. K.: Incremental evaluation of computational circuits. In: *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 1990, S. 32–42

[2] Alpert, S. R.: Graceful Interaction with Graphical Constraints. In: *IEEE Comput. Graph. Appl.* 13 (1993), Nr. 2, S. 82–91

[3] Anderson, R. J. ; Sobti, S. : The table layout problem. In: *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*. New York, NY, USA : ACM Press, 1999, S. 115–123

[4] Apt, K. R.: The essence of constraint propagation. In: *Theoretical Computer Science* 221 (1999), Nr. 1–2, S. 179–210

[5] Apt, K. R.: The Rough Guide to Constraint Propagation. In: *Principles and Practice of Constraint Programming*, 1999, S. 1–23

[6] Badros, G. : *Extending Interactive Graphical Applications with Constraints*. 2000

[7] Badros, G. J. ; Borning, A. ; Stuckey, P. J.: The Cassowary linear arithmetic constraint solving algorithm. In: *ACM Trans. Comput.-Hum. Interact.* 8 (2001), Nr. 4, S. 267–306

[8] Badros, G. J. ; Tirtowidjojo, J. J. ; Marriott, K. ; Meyer, B. ; Portnoy, W. ; Borning, A. : A constraint extension to scalable vector graphics. In: *WWW '01: Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA : ACM Press, 2001, S. 489–498

[9] Batini, C. ; Nardelli, E. ; Tamassia, R. : A Layout algorithm for data flow diagrams. In: *IEEE Trans. Softw. Eng.* 12 (1986), Nr. 4, S. 538–546

[10] Baudisch, P. ; Cutrell, E. ; Hinckley, K. ; Eversole, A. : Snap-and-go: helping users align objects without the modality of traditional snapping. In: *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 2005, S. 301–310

[11] Behringer, F. A.: On optimal decisions under complete ignorance : A new criterion stronger than both Pareto and Maxmin. In: *European Journal of Operational Research* 1 (1977), Nr. 5, S. 295 – 306

[12] Bentley, J. L. ; Ottmann., T. : Algorithms for reporting and counting geometric intersections. In: *IEEE Trans. Comput.* C28 (1979), Nr. 9, S. 643–647

[13] Bier, E. A. ; Stone, M. C.: Snap-dragging. In: *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1986, S. 233–240

[14] Bolz, D. : Some aspects of the user interface of a knowledge based beautifier for drawings. In: *IUI '93: Proceedings of the 1st international conference on Intelligent user interfaces*. New York, NY, USA : ACM Press, 1993, S. 45–52

[15] Borning, A. ; Anderson, R. ; Freeman-Benson, B. : Indigo: a local propagation algorithm for inequality constraints. In: *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 1996, S. 129–136

[16] Borning, A. ; Duisberg, R. : Constraint-based tools for building user interfaces. In: *ACM Trans. Graph.* 5 (1986), Nr. 4, S. 345–374

[17] Borning, A. ; Duisberg, R. ; Freeman-Benson, B. ; Kramer, A. ; Woolf, M. : Constraint hierarchies. In: *SIGPLAN Not.* 22 (1987), Nr. 12, S. 48–60

[18] Borning, A. ; Marriott, K. ; Stuckey, P. ; Xiao, Y. : Solving linear arithmetic constraints for user interface applications. In: *UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 1997, S. 87–96

[19] Borning, A. ; Martindale, A. ; Wilson, M. ; Maher, M. : Constraint hierarchies and logic programming. In: *In Proceedings of*

the Sixth International Conference on Logic Programming, MIT Press, 1989, S. 149–164

[20] Borning, A. H.: *Thinglab–a constraint-oriented simulation laboratory.* Stanford, CA, USA, Stanford University, Diss., 1979

[21] Bouma, W. ; Fudos, I. ; Hoffmann, C. ; Cai, J. ; Paige, R. : Geometric constraint solver. In: *Computer-aided Design* 27 (1995), Nr. 6, S. 487–501

[22] Carreira-Perpiñán, M. A.: *"Simplex Method." From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein.* 2009

[23] Carrizosa, E. ; Fliege, J. : Generalized Goal Programming: polynomial methods and applications. In: *Mathematical Programming* 93 (2002), Nr. 2, S. 281–303

[24] Chok, S. S. ; Marriott, K. : Automatic construction of intelligent diagram editors. In: *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology.* New York, NY, USA : ACM Press, 1998, S. 185–194

[25] COIN-OR Project: *CLP Linear Programming Solver.* 2009

[26] COIN-OR Project: *DyLP Dynamic Simplex Solver.* 2009

[27] COIN-OR Project: *OSI: Open Solver Interface.* 2009

[28] *Kapitel* 29 (Linear Programming). In: Cormen, T. H. ; Leiserson, C. E. ; Rivest, R. L. ; Stein, C. : *Introduction to Algorithms.* Second Edition. MIT Press and McGraw-Hill, 2001

[29] Dantzig, G. B.: *Linear Programming and Extensions.* Princeton, NJ: Princeton University Press, 1963

[30] Dantzig, G. B. ; Thapa, M. N.: *Linear Programming 2: Theory and Extensions.* Springer-Verlag, 2003

[31] Demers, A. ; Reps, T. ; Teitelbaum, T. : Incremental evaluation for attribute grammars with application to syntax-directed editors. In: *POPL '81: Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* New York, NY, USA : ACM, 1981, S. 105–116

[32] Di Iorio, A. ; Furini, L. ; Vitali, F. ; Lumley, J. ; Wiley, T. : Higher-level layout through topological abstraction. In: *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*. New York, NY, USA : ACM, 2008, S. 90–99

[33] Ehrgott, M. : A characterization of lexicographic max-ordering solutions / University of Kaiserslautern, Department of Mathematics. 1996 (12). – Forschungsbericht

[34] Eichelberger, H. ; Gudenberg, J. W.: *UML Class Diagrams - State of the Art in Layout Techniques*. 2003

[35] Eiglsperger, M. ; Gutwenger, C. ; Kaufmann, M. ; Kupke, J. ; Jünger, M. ; Leipert, S. ; Klein, K. ; Mutzel, P. ; Siebenhaller, M. : Automatic layout of UML class diagrams in orthogonal style. In: *Information Visualization* 3 (2004), Nr. 3, S. 189–208

[36] Feiner, S. K.: A grid-based approach to automating display layout. In: *Proceedings on Graphics interface '88*. Toronto, Ont., Canada, Canada : Canadian Information Processing Society, 1988, S. 192–197

[37] Flavell, R. : A new goal programming formulation. In: *Omega* 4 (1976), Nr. 6, S. 731 – 732

[38] Fogarty, J. ; Hudson, S. E.: GADGET: a toolkit for optimization-based approaches to interface and display generation. In: *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 2003, S. 125–134

[39] Freeman-Benson, B. N. ; Maloney, J. ; Borning, A. : An incremental constraint solver. In: *Commun. ACM* 33 (1990), Nr. 1, S. 54–63

[40] Fudos, I. : *Geometric constraint solving*, Department of Computer Sciences, Purdue University, Diss., 1995

[41] Fudos, I. ; Hoffmann, C. M.: A graph-constructive approach to solving systems of geometric constraints. In: *ACM Trans. Graph.* 16 (1997), Nr. 2, S. 179–216

[42] Gajos, K. ; Weld, D. S.: SUPPLE: automatically generating user interfaces. In: *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*. New York, NY, USA : ACM Press, 2004, S. 93–100

[43] Gal, T. : On efficient sets in vector maximum problems – A brief survey. In: *European Journal of Operational Research* 24 (1986), Nr. 2, S. 253 – 264

[44] Gleicher, M. ; Witkin, A. : Drawing with Constraints. In: *The Visual Computer* 11 (1994), Nr. 1

[45] Gleicher, M. : Briar: a constraint-based drawing program. In: *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1992, S. 661–662

[46] Gleicher, M. : Integrating constraints and direct manipulation. In: *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*. New York, NY, USA : ACM Press, 1992, S. 171–174

[47] Gleicher, M. : Practical Issues in Graphical Constraints. In: *Principles and Practice of Constraint Programming*, 1993, S. 98–106

[48] Gleicher, M. : Retargetting motion to new characters. In: *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1998, S. 33–42

[49] Gleicher, M. ; Litwinowicz, P. : Constraint-based motion adaptation. In: *The Journal of Visualization and Computer Animation* 9 (1998), Nr. 2, S. 65–94

[50] Graf, W. ; Kroener, A. ; Neurohr, S. ; Goebel, R. : Experience in Integrating AI and Constraint Programming – Methods for Automated Yellow Pages Layout. In: *KI* 12 (1998), Nr. 2, S. 79–85

[51] Graf, W. ; Neurohr, S. : Constraint-Based Layout in Visual Program Design. In: *Visual Languages*, 1995, S. 116–117

[52] Graf, W. H.: The constraint-based layout framework LayLab and its applications. In: *Proceedings of ACM Workshop on Effective Abstractions in Multimedia, Layout and Interaction*, 1995

[53] Gutwenger, C. ; Jünger, M. ; Klein, K. ; Kupke, J. ; Leipert, S. ; Mutzel, P. : A new approach for visualizing UML class diagrams. In: *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*. New York, NY, USA : ACM Press, 2003, S. 179–188

[54] Harrington, S. J. ; Naveda, J. F. ; Jones, R. P. ; Roetling, P. ; Thakkar, N. : Aesthetic measures for automated document layout. In: *DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering*. New York, NY, USA : ACM Press, 2004, S. 109–111

[55] Helm, R. ; Huynh, T. ; Marriott, K. ; Vlissides, J. M.: An Object-Oriented Architecture for Constraint-Based Graphical Editing. In: *Workshops on Object-Oriented Graphics (2)*, 1992, S. 217–238

[56] Heydon, A. ; Nelson, G. : The Juno-2 constraint-based drawing editor / Digital Systems Research Center. 1994 (131a). – Tech. Rep.

[57] Hill, R. D.: The Rendezvous constraint maintenance system. In: *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 1993, S. 225–234

[58] Hoover, R. : *Incremental graph evaluation (attribute grammar)*. Ithaca, NY, USA, Cornell University, Diss., 1987

[59] Hosobe, H. : A Scalable Linear Constraint Solver for User Interface Construction. In: *Principles and Practice of Constraint Programming –CP 2000* (2000), S. 218–233

[60] Hosobe, H. : A modular geometric constraint solver for user interface applications. In: *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 2001, S. 91–100

[61] Hosobe, H. : Hierarchical nonlinear constraint satisfaction. In: *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*. New York, NY, USA : ACM Press, 2004, S. 16–20

[62] Hosobe, H. ; Matsuoka, S. : A Foundation of Solution Methods for Constraint Hierarchies. In: *Constraints* 8 (2003), Nr. 1, S. 41–59

[63] Hower, W. ; Graf, W. H.: Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey / Deutsches Forschungszentrum für Künstliche Intelligenz GmbH. 1995 (RR-95-12). – Forschungsbericht

[64] Hower, W. ; Graf, W. H.: A bibliographical survey of constraint-based approaches to CAD, graphics, layout, visualization, and related

topics. In: *Knowledge-Based Systems* 9 (1996), November, Nr. 7, S. 449–464

[65] Hudson, S. E.: User interface specification using an enhanced spreadsheet model. In: *ACM Trans. Graph.* 13 (1994), Nr. 3, S. 209–239

[66] Hudson, S. E. ; Mohamed, S. P.: Interactive specification of flexible user interface displays. In: *ACM Trans. Inf. Syst.* 8 (1990), Nr. 3, S. 269–288

[67] Hurst, N. ; Marriott, K. : Approximating text by its area. In: *DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering*. New York, NY, USA : ACM, 2007, S. 147–150

[68] Hurst, N. ; Marriott, K. : Satisficing scrolls: a shortcut to satisfactory layout. In: *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*. New York, NY, USA : ACM, 2008, S. 131–140

[69] Hurst, N. ; Marriott, K. ; Albrecht, D. : Solving the simple continuous table layout problem. In: *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*. New York, NY, USA : ACM, 2006, S. 28–30

[70] Hurst, N. ; Marriott, K. ; Moulder, P. : Dynamic approximation of complex graphical constraints by linear constraints. In: *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 2002, S. 191–200

[71] Hurst, N. ; Marriott, K. ; Moulder, P. : Toward tighter tables. In: *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*. New York, NY, USA : ACM Press, 2005, S. 74–83

[72] Hutchins, E. L. ; Hollan, J. D. ; Norman, D. A.: Direct manipulation interfaces. In: *Hum.-Comput. Interact.* 1 (1985), Nr. 4, S. 311–338

[73] Igarashi, T. ; Matsuoka, S. ; Kawachiya, S. ; Tanaka, H. : Interactive beautification: a technique for rapid geometric design. In: *UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 1997, S. 105–114

[74] ILOG SA: *ILOG Optimization Algorithms*. 2009

[75] Jacobs, C. ; Li, W. ; Schrier, E. ; Bargeron, D. ; Salesin, D. : Adaptive document layout. In: *Commun. ACM* 47 (2004), Nr. 8, S. 60–66

[76] Jacobs, C. ; Li, W. ; Schrier, E. ; Bargeron, D. ; Salesin, D. : Adaptive grid-based document layout. In: *ACM Trans. Graph.* 22 (2003), Nr. 3, S. 838–847

[77] Johari, R. ; Marks, J. ; Partovi, A. ; Shieber, S. : Automatic Yellow-Pages pagination and layout. In: *Journal of Heuristics* 2 (1997), Nr. 4, S. 321–342

[78] Karmarkar, N. : A new polynomial-time algorithm for linear programming. In: *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*. New York, NY, USA : ACM, 1984, S. 302–311

[79] Karsenty, S. ; Landay, J. A. ; Weikart, C. : Inferring graphical constraints with Rockit. In: *HCI'92: Proceedings of the conference on People and computers VII*. New York, NY, USA : Cambridge University Press, 1993, S. 137–153

[80] Klee, V. ; Minty, G. J.: How good is the simplex algorithm? In: Shisha, O. (Hrsg.): *Inequalities* Bd. III. New York : Academic Press, 1972, S. 159–175

[81] Knuth, D. E. ; Plass, M. F.: Breaking paragraphs into lines. In: *Software: Practice and Experience* 11 (1981), Nr. 11, S. 1119–1184

[82] Kornbluth, J. : A survey of goal programming. In: *Omega* 1 (1973), Nr. 2, S. 193 – 205

[83] Kramer, G. A.: A geometric constraint engine. In: *Artif. Intell.* 58 (1992), Nr. 1-3, S. 327–360

[84] Lin, X. : Active layout engine: Algorithms and applications in variable data printing. In: *Computer-Aided Design* 38 (2006), May, Nr. 5, S. 444–456

[85] Lok, S. ; Feiner, S. ; Ngai, G. : Evaluation of visual balance for automated layout. In: *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*. New York, NY, USA : ACM Press, 2004, S. 101–108

[86] Lutteroth, C. ; Strandh, R. ; Weber, G. : Domain Specific High-Level Constraints for User Interface Layout. In: *Constraints* (2008)

[87] Lutteroth, C. ; Weber, G. : User Interface Layout with Ordinal and Linear Constraints. In: *Proceedings of the Seventh Australasian User Interface Conference (AUIC 2006)*, 2006, S. 53–60

[88] Mackinlay, J. D.: *Automatic design of graphical presentations*. Stanford, CA, USA, Stanford University, Diss., 1987

[89] Mackinlay, J. : Automating the design of graphical presentations of relational information. In: *ACM Trans. Graph.* 5 (1986), Nr. 2, S. 110–141

[90] Marriott, K. ; Chok, S. S.: QOCA: A constraint solving toolkit for interactive graphical applications. In: *Constraints* 7 (2002), Nr. 3/4, S. 229–254

[91] Marriott, K. ; Chok, S. C. ; Finlay, A. : A Tableau Based Constraint Solving Toolkit for Interactive Graphical Applications. In: *CP '98: Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*. London, UK : Springer-Verlag, 1998, S. 340–354

[92] Marriott, K. ; Moulder, P. ; Hurst, N. : Automatic float placement in multi-column documents. In: *DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering*. New York, NY, USA : ACM, 2007, S. 125–134

[93] Marriott, K. ; Moulder, P. ; Stuckey, P. J. ; Borning, A. : Solving Disjunctive Constraints for Interactive Graphical Applications. In: *Lecture Notes in Computer Science* 2239 (2001), S. 361+

[94] McCormack, C. ; Marriott, K. ; Meyer, B. : Authoring adaptive diagrams. In: *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*. New York, NY, USA : ACM, 2008, S. 154–163

[95] MOSEK ApS: *Mosek Optimization Toolkit*. 2009

[96] Myers, B. ; McDaniel, R. ; Miller, R. : *The Amulet Prototype-Instance Framework*. 1999

[97] Myers, B. A.: Creating user interfaces using programming by example, visual programming, and constraints. In: *ACM Trans. Program. Lang. Syst.* 12 (1990), Nr. 2, S. 143–177

[98] Myers, B. A. ; Giuse, D. A. ; Dannenberg, R. B. ; Kosbie, D. S. ; Pervin, E. ; Mickish, A. ; Zanden, B. V. ; Marchal, P. : Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces. In: *Computer* 23 (1990), Nr. 11, S. 71–85

[99] Myers, B. A. ; McDaniel, R. G. ; Miller, R. C. ; Ferrency, A. S. ; Faulring, A. ; Kyle, B. D. ; Mickish, A. ; Klimovitski, A. ; Doane, P. : The Amulet Environment: New Models for Effective User Interface Software Development. In: *Software Engineering* 23 (1997), Nr. 6, S. 347–365

[100] Nelson, G. : Juno, a constraint-based graphics system. In: *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1985, S. 235–243

[101] Ogryczak, W. ; Sliwinski, T. : *Lexicographic Max-Min Optimization for Efficient and Fair Bandwidth Allocation.* International Network Optimization Conference (http://www.euro-online.org/enog/inoc2007), April 2007

[102] Oliveira, a. B. S. Jo Two algorithms for automatic document page layout. In: *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*. New York, NY, USA : ACM, 2008, S. 141–149

[103] Pioro, M. ; Dzida, M. ; Kubilinskas, E. ; Nilsson, P. : Applications of the max-min fairness principle in telecommunication network design. In: *Next Generation Internet Networks*, 2005, S. 219–225

[104] Purvis, L. ; Harrington, S. ; O'Sullivan, B. ; Freuder, E. C.: Creating personalized documents: an optimization approach. In: *DocEng '03: Proceedings of the 2003 ACM symposium on Document engineering*. New York, NY, USA : ACM Press, 2003, S. 68–77

[105] Reps, T. ; Teitelbaum, T. ; Demers, A. : Incremental Context-Dependent Analysis for Language-Based Editors. In: *ACM Trans. Program. Lang. Syst.* 5 (1983), Nr. 3, S. 449–477

[106] Romero, C. ; Tamiz, M. ; Jones, D. F.: Goal Programming, Compromise Programming and Reference Point Method Formulations: Linkages and Utility Interpretations. In: *The Journal of the Operational Research Society* 49 (1998), Nr. 9, S. 986–991

[107] Sannella, M. ; Maloney, J. ; Freeman-Benson, B. N. ; Borning, A. : Multi-way versus One-way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm. In: *Software - Practice and Experience* 23 (1993), Nr. 5, S. 529–566

[108] Seybold, C. ; Glinz, M. ; Meier, S. ; Merlo-Schett, N. : An effective layout adaptation technique for a graphical modeling tool. In: *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2003, S. 826–827

[109] Shneiderman, B. : The future of interactive systems and the emergence of direct manipulation. In: *Behaviour & Information Technology* 1 (1982), Nr. 3, S. 237 – 256

[110] Shneiderman, B. : Direct manipulation for comprehensible, predictable and controllable user interfaces. In: *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*. New York, NY, USA : ACM Press, 1997, S. 33–39

[111] Smith, J. A. ; Linders, J. G.: Automatic generation of logic diagrams. In: *DAC '76: Proceedings of the 13th conference on Design automation*. New York, NY, USA : ACM Press, 1976, S. 377–391

[112] Stephen, T. ; Tuncel, L. ; Luss, H. : On Equitable Resource Allocation Problems: a Lexicographic Minimax Approach. In: *Oper. Res.* 47 (1999), Nr. 3, S. 361–376

[113] Sutherland, I. E.: Sketch pad a man-machine graphical communication system. In: *DAC '64: Proceedings of the SHARE design automation workshop*. New York, NY, USA : ACM Press, 1964, S. 6.329–6.346

[114] Tamiz, M. ; Jones, D. ; Romero, C. : Goal programming for decision making: An overview of the current state-of-the-art. In: *European Journal of Operational Research* 111 (1998), Nr. 3, S. 569 – 581

[115] Wang, X. ; Wood, D. : Tabular formatting problems. In: *In 3rd Principles of Document Processing* (1996), S. 171–181

[116] Weitzman, L. ; Wittenburg, K. : Automatic presentation of multimedia documents using relational grammars. In: *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*. New York, NY, USA : ACM Press, 1994, S. 443–451

[117] Wybrow, M. ; Marriott, K. ; Mciver, L. ; Stuckey, P. J.: Comparing usability of one-way and multi-way constraints for diagram editing. In: *ACM Trans. Comput.-Hum. Interact.* 14 (2008), Nr. 4, S. 1–38

[118] Zanden, B. T. V.: An incremental algorithm for satisfying hierarchies of multi-way dataflow constraints. In: *ACM Trans. Program. Lang. Syst.* 18 (1996), Jan, Nr. 1, S. 30 – 72

[119] Zanden, B. T. V. ; Halterman, R. ; Myers, B. A. ; McDaniel, R. ; Miller, R. ; Szekely, P. ; Giuse, D. A. ; Kosbie, D. : Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits. In: *ACM Trans. Program. Lang. Syst.* 23 (2001), Nr. 6, S. 776–796

[120] Zanden, B. T. V. ; Venckus, S. A.: An empirical study of constraint usage in graphical applications. In: *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 1996, S. 137–146

Appendix A

A Complete Constraint System

On the following pages, the constraints generated to create the layout shown in Fig. A.1 are reproduced in full. The layout has been created with the latest version of the ICBM system, which explains the slight differences compared to Fig. 6.4. As explained in chapter 6 on page 126, the text boxes are represented using four inner and four outer constraints.

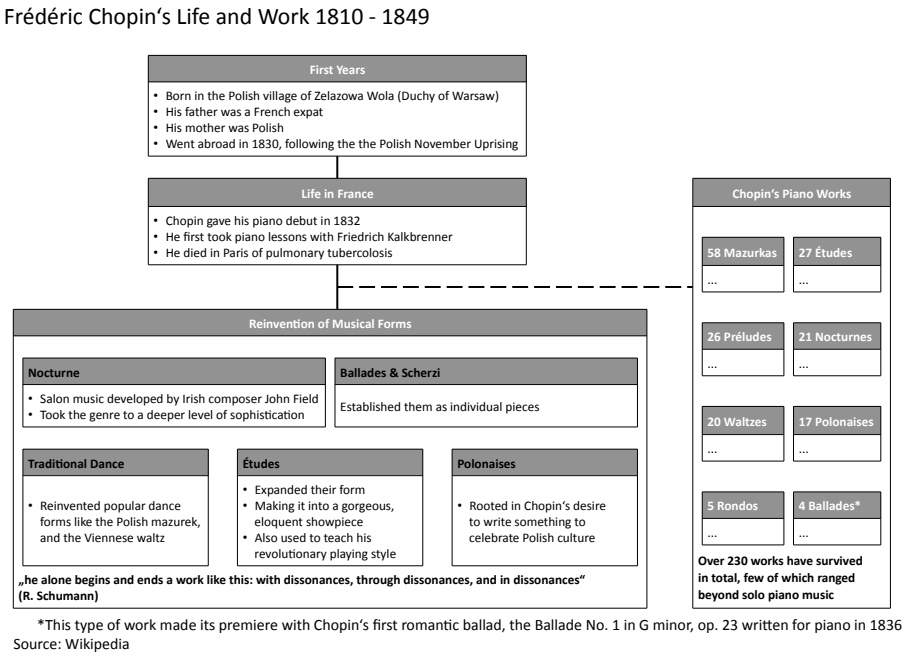


Figure A.1: The familiar slide sample, shown again.

$$
\begin{aligned}
-x_1 + x_2 - x_3 &= 0 \\
-x_4 + x_5 - x_3 &= 0 \\
-x_4 + x_6 - x_7 &= 0 \\
-x_8 + x_2 - x_7 &= 0 \\
-x_9 + x_{10} - x_7 &= 0 \\
-x_{11} + x_{12} &= 320
\end{aligned}
$$

Textbox: First Years

$$
\begin{aligned}
-x_{16} + x_{19} &\geq 96 \\
-1.51894x_{16} + 1.51894x_{19} - x_{14} + x_{17} &\geq 643.636 \\
-x_{14} + x_{17} &\geq 189
\end{aligned}
$$

Textbox: Born in the Polish village of ...

$$
\begin{aligned}
-x_{22} + x_{24} &\geq 384 \\
-4.80632x_{22} + 4.80632x_{24} - x_{21} + x_{23} &\geq 4869.44 \\
-1.10654x_{22} + 1.10654x_{24} - x_{21} + x_{23} &\geq 2024.82 \\
-0.641207x_{22} + 0.641207x_{24} - x_{21} + x_{23} &\geq 1552.78 \\
-0.187134x_{22} + 0.187134x_{24} - x_{21} + x_{23} &\geq 872.263 \\
-0.14625x_{22} + 0.14625x_{24} - x_{21} + x_{23} &\geq 777.799 \\
-x_{21} + x_{23} &\geq 425
\end{aligned}
$$

Textbox: Life in France

$$
\begin{aligned}
-x_{28} + x_{30} &\geq 96 \\
-1.91968x_{28} + 1.91968x_{30} - x_{26} + x_{29} &\geq 814.578 \\
-x_{26} + x_{29} &\geq 228
\end{aligned}
$$

Textbox: Chopin gave his piano debut in 1832 ...

$$
\begin{aligned}
-x_{33} + x_{35} &\geq 288 \\
-2.95415x_{33} + 2.95415x_{35} - x_{32} + x_{34} &\geq 2950.39 \\
-2.60905x_{33} + 2.60905x_{35} - x_{32} + x_{34} &\geq 2762.81
\end{aligned}
$$

$$
\begin{aligned}
-0.959559x_{33} + 0.959559x_{35} - x_{32} + x_{34} &\geq 1655.18 \\
-0.366269x_{33} + 0.366269x_{35} - x_{32} + x_{34} &\geq 987.104 \\
-0.0161952x_{33} + 0.0161952x_{35} - x_{32} + x_{34} &\geq 499.766 \\
-x_{32} + x_{34} &\geq 477
\end{aligned}
$$

Textbox: Reinvention of Musical Forms

$$
\begin{aligned}
-x_{40} + x_{43} &\geq 96 \\
-3.8541x_{40} + 3.8541x_{43} - x_{38} + x_{41} &\geq 1734.99 \\
-0.917214x_{40} + 0.917214x_{43} - x_{38} + x_{41} &\geq 761.158 \\
-x_{38} + x_{41} &\geq 411
\end{aligned}
$$

Textbox: Nocturne

$$
\begin{aligned}
-x_{47} + x_{49} &\geq 96 \\
-x_{45} + x_{48} &\geq 319
\end{aligned}
$$

Textbox: Salon music developed by Irish composer John Field ...

$$
\begin{aligned}
-x_{52} + x_{54} &\geq 192 \\
-3.64856x_{52} + 3.64856x_{54} - x_{51} + x_{53} &\geq 3091.05 \\
-1.1583x_{52} + 1.1583x_{54} - x_{51} + x_{53} &\geq 1588.18 \\
-0.231086x_{52} + 0.231086x_{54} - x_{51} + x_{53} &\geq 782.843 \\
-x_{51} + x_{53} &\geq 533
\end{aligned}
$$

Textbox: Ballades & Scherzi

$$
\begin{aligned}
-x_{57} + x_{59} &\geq 96 \\
-2.47002x_{57} + 2.47002x_{59} - x_{56} + x_{58} &\geq 1086.24 \\
-0.392471x_{57} + 0.392471x_{59} - x_{56} + x_{58} &\geq 429.032 \\
-x_{56} + x_{58} &\geq 286
\end{aligned}
$$

Textbox: Established them as individual pieces

$$
\begin{aligned}
-x_{61} + x_{63} &\geq 96 \\
-4.38803x_{61} + 4.38803x_{63} - x_{60} + x_{62} &\geq 2051.5 \\
-1.35526x_{61} + 1.35526x_{63} - x_{60} + x_{62} &\geq 1065.42
\end{aligned}
$$

$$-x_{60} + x_{62} \geq 376$$

Textbox: Traditional Dance

$$\begin{aligned} -x_{66} + x_{68} &\geq 96 \\ -1.95546x_{66} + 1.95546x_{68} - x_{64} + x_{67} &\geq 969.449 \\ -x_{64} + x_{67} &\geq 371 \end{aligned}$$

Textbox: Reinvented popular dance forms ...

$$\begin{aligned} -x_{71} + x_{73} &\geq 96 \\ -9.22813x_{71} + 9.22813x_{73} - x_{70} + x_{72} &\geq 4471.8 \\ -2.97015x_{71} + 2.97015x_{73} - x_{70} + x_{72} &\geq 2254.4 \\ -0.832348x_{71} + 0.832348x_{73} - x_{70} + x_{72} &\geq 1200.43 \\ -0.487598x_{71} + 0.487598x_{73} - x_{70} + x_{72} &\geq 942.285 \\ -x_{70} + x_{72} &\geq 451 \end{aligned}$$

Textbox: Polonaises

$$\begin{aligned} -x_{76} + x_{78} &\geq 96 \\ -x_{75} + x_{77} &\geq 363 \end{aligned}$$

Textbox: Rooted in Chopin's desire to write ...

$$\begin{aligned} -x_{80} + x_{82} &\geq 96 \\ -8.0358x_{80} + 8.0358x_{82} - x_{79} + x_{81} &\geq 3960.87 \\ -3.09139x_{80} + 3.09139x_{82} - x_{79} + x_{81} &\geq 2186.32 \\ -1.29602x_{80} + 1.29602x_{82} - x_{79} + x_{81} &\geq 1459.09 \\ -0.491994x_{80} + 0.491994x_{82} - x_{79} + x_{81} &\geq 900.083 \\ -0.473757x_{80} + 0.473757x_{82} - x_{79} + x_{81} &\geq 885.846 \\ -x_{79} + x_{81} &\geq 425 \end{aligned}$$

Textbox: Études

$$-x_{84} + x_{86} \geq 96$$

$$-x_{83} + x_{85} \geq 233$$

Textbox: Expanded their form

$$\begin{aligned}
-x_{88} + x_{90} &\geq 288 \\
-3.28924x_{88} + 3.28924x_{90} - x_{87} + x_{89} &\geq 3179.84 \\
-0.995381x_{88} + 0.995381x_{90} - x_{87} + x_{89} &\geq 1594.9 \\
-0.81532x_{88} + 0.81532x_{90} - x_{87} + x_{89} &\geq 1459.71 \\
-x_{87} + x_{89} &\geq 518
\end{aligned}$$

Textbox: Chopin's Piano Works

$$\begin{aligned}
-x_{93} + x_{96} &\geq 106 \\
-2.33186x_{93} + 2.33186x_{96} - x_{92} + x_{94} &\geq 1227.35 \\
-1.19529x_{93} + 1.19529x_{96} - x_{92} + x_{94} &\geq 808.104 \\
-x_{92} + x_{94} &\geq 295
\end{aligned}$$

Textbox: 58 Mazurkas

$$\begin{aligned}
-x_{100} + x_{103} &\geq 96 \\
-x_{98} + x_{101} &\geq 429
\end{aligned}$$

Textbox: ...

$$\begin{aligned}
-x_{106} + x_{108} &\geq 96 \\
-x_{105} + x_{107} &\geq 68
\end{aligned}$$

Textbox: 26 Préludes

$$\begin{aligned}
-x_{112} + x_{114} &\geq 96 \\
-x_{110} + x_{113} &\geq 397
\end{aligned}$$

Textbox: ...

$$\begin{aligned}
-x_{117} + x_{119} &\geq 96 \\
-x_{116} + x_{118} &\geq 68
\end{aligned}$$

Textbox: 20 Waltzes

$$-x_{123} + x_{125} \geq 96$$
$$-x_{121} + x_{124} \geq 372$$

Textbox: ...

$$-x_{128} + x_{130} \geq 96$$
$$-x_{127} + x_{129} \geq 68$$

Textbox: 5 Rondos

$$-x_{134} + x_{136} \geq 96$$
$$-x_{132} + x_{135} \geq 316$$

Textbox: ...

$$-x_{139} + x_{141} \geq 96$$
$$-x_{138} + x_{140} \geq 68$$

Textbox: 27 Études

$$-x_{145} + x_{148} \geq 106$$
$$-x_{144} + x_{146} \geq 333$$

Textbox: ...

$$-x_{150} + x_{152} \geq 96$$
$$-x_{149} + x_{151} \geq 68$$

Textbox: 21 Nocturnes

$$-x_{154} + x_{156} \geq 96$$
$$-x_{153} + x_{155} \geq 451$$

Textbox: ...

$$-x_{158} + x_{160} \geq 96$$
$$-x_{157} + x_{159} \geq 68$$

Textbox: 17 Polonaises

$$-x_{162} + x_{164} \geq 106$$
$$-x_{161} + x_{163} \geq 463$$

Textbox: ...

$$-x_{166} + x_{168} \geq 96$$
$$-x_{165} + x_{167} \geq 68$$

Textbox: 4 Ballades*

$$-x_{170} + x_{172} \geq 96$$
$$-x_{169} + x_{171} \geq 385$$

Textbox: ...

$$-x_{174} + x_{176} \geq 96$$
$$-x_{173} + x_{175} \geq 68$$

Textbox: Over 230 works have survived in total ...

$$-x_{180} + x_{183} \geq 106$$
$$-8.96571x_{180} + 8.96571x_{183} - x_{178} + x_{181} \geq 4720.73$$
$$-3.05906x_{180} + 3.05906x_{183} - x_{178} + x_{181} \geq 2399.78$$

$$
\begin{aligned}
-1.71227x_{180} + 1.71227x_{183} - x_{178} + x_{181} &\geq 1732 \\
-1.41061x_{180} + 1.41061x_{183} - x_{178} + x_{181} &\geq 1544.62 \\
-0.992583x_{180} + 0.992583x_{183} - x_{178} + x_{181} &\geq 1264.28 \\
-0.737385x_{180} + 0.737385x_{183} - x_{178} + x_{181} &\geq 1075.14 \\
-0.282372x_{180} + 0.282372x_{183} - x_{178} + x_{181} &\geq 711.177 \\
-0.246329x_{180} + 0.246329x_{183} - x_{178} + x_{181} &\geq 658.22 \\
-x_{178} + x_{181} &\geq 290
\end{aligned}
$$

Textbox: “he alone begins and ends a work like this: ...

$$
\begin{aligned}
-x_{188} + x_{191} &\geq 192 \\
-11.2439x_{188} + 11.2439x_{191} - x_{186} + x_{189} &\geq 6671.24 \\
-4.86606x_{188} + 4.86606x_{191} - x_{186} + x_{189} &\geq 3697.57 \\
-2.33384x_{188} + 2.33384x_{191} - x_{186} + x_{189} &\geq 2428.29 \\
-0.786047x_{188} + 0.786047x_{191} - x_{186} + x_{189} &\geq 1312.68 \\
-0.508548x_{188} + 0.508548x_{191} - x_{186} + x_{189} &\geq 1051.21 \\
-x_{186} + x_{189} &\geq 489
\end{aligned}
$$

Textbox: *This type of work made its premiere ...

$$
\begin{aligned}
-x_{194} + x_{197} &\geq 364 \\
-14.8468x_{194} + 14.8468x_{197} - x_{193} + x_{195} &\geq 12492.6 \\
-4.67373x_{194} + 4.67373x_{197} - x_{193} + x_{195} &\geq 5586.2 \\
-3.10628x_{194} + 3.10628x_{197} - x_{193} + x_{195} &\geq 4270.35 \\
-1.7066x_{194} + 1.7066x_{197} - x_{193} + x_{195} &\geq 3009.24 \\
-0.910062x_{194} + 0.910062x_{197} - x_{193} + x_{195} &\geq 2038.86 \\
-0.35231x_{194} + 0.35231x_{197} - x_{193} + x_{195} &\geq 1336.91 \\
-0.310102x_{194} + 0.310102x_{197} - x_{193} + x_{195} &\geq 1278.49 \\
-x_{193} + x_{195} &\geq 704
\end{aligned}
$$

Textbox: Frédéric Chopin’s Life and Work 1810 - 1849

$$
\begin{aligned}
-x_{199} + x_{201} &\geq 307 \\
-5.29555x_{199} + 5.29555x_{201} - x_{198} + x_{200} &\geq 5126.86
\end{aligned}
$$

$$-1.48351x_{199} + 1.48351x_{201} - x_{198} + x_{200} \geq 2270.73$$
$$-0.744977x_{199} + 0.744977x_{201} - x_{198} + x_{200} \geq 1519.11$$
$$-x_{198} + x_{200} \geq 508$$

Horizontal Gap Constraints

$$x_{198} \geq 0 \qquad x_{37} \geq 0 \qquad x_{193} \geq 0$$
$$-x_{37} + x_{186} \geq 0 \qquad -x_{37} + x_{4} \geq 0 \qquad -x_{4} + x_{45} \geq 0$$
$$-x_{4} + x_{64} \geq 0 \qquad -x_{4} + x_{51} \geq 0 \qquad -x_{4} + x_{70} \geq 0$$
$$-x_{45} + x_{48} \geq 0 \qquad -x_{64} + x_{67} \geq 0 \qquad x_{13} \geq 0$$
$$-x_{13} + x_{32} \geq 0 \qquad -x_{13} + x_{21} \geq 0 \qquad -x_{70} + x_{72} \geq 0$$
$$-x_{72} + x_{6} \geq 0 \qquad -x_{67} + x_{6} \geq 0 \qquad -x_{6} + x_{9} \geq 0$$
$$-x_{9} + x_{83} \geq 0 \qquad -x_{9} + x_{87} \geq 0 \qquad -x_{37} + x_{38} \geq 0$$
$$-x_{83} + x_{85} \geq 0 \qquad -x_{13} + x_{26} \geq 0 \qquad -x_{13} + x_{14} \geq 0$$
$$-x_{51} + x_{53} \geq 0 \qquad -x_{53} + x_{5} \geq 0 \qquad -x_{48} + x_{5} \geq 0$$
$$-x_{5} + x_{1} \geq 0 \qquad -x_{13} + x_{202} \geq 0 \qquad -x_{37} + x_{202} \geq 0$$
$$-x_{1} + x_{56} \geq 0 \qquad -x_{1} + x_{60} \geq 0 \qquad -x_{14} + x_{17} \geq 0$$
$$-x_{26} + x_{29} \geq 0 \qquad -x_{87} + x_{89} \geq 0 \qquad -x_{38} + x_{41} \geq 0$$
$$-x_{89} + x_{10} \geq 0 \qquad -x_{85} + x_{10} \geq 0 \qquad -x_{198} + x_{200} \geq 0$$
$$-x_{56} + x_{58} \geq 0 \qquad -x_{10} + x_{8} \geq 0 \qquad -x_{32} + x_{34} \geq 0$$
$$-x_{8} + x_{75} \geq 0 \qquad -x_{8} + x_{79} \geq 0 \qquad -x_{75} + x_{77} \geq 0$$
$$-x_{21} + x_{23} \geq 0 \qquad -x_{23} + x_{18} \geq 0 \qquad -x_{34} + x_{18} \geq 0$$
$$-x_{29} + x_{18} \geq 0 \qquad -x_{17} + x_{18} \geq 0 \qquad -x_{202} + x_{18} \geq 0$$
$$-x_{60} + x_{62} \geq 0 \qquad -x_{186} + x_{189} \geq 0 \qquad -x_{79} + x_{81} \geq 0$$
$$-x_{81} + x_{2} \geq 0 \qquad -x_{62} + x_{2} \geq 0 \qquad -x_{77} + x_{2} \geq 0$$
$$-x_{58} + x_{2} \geq 0 \qquad -x_{2} + x_{42} \geq 0 \qquad -x_{189} + x_{42} \geq 0$$
$$-x_{41} + x_{42} \geq 0 \qquad -x_{202} + x_{42} \geq 0 \qquad -x_{42} + x_{91} \geq 0$$
$$-x_{18} + x_{91} \geq 0 \qquad -x_{91} + x_{178} \geq 0 \qquad -x_{91} + x_{97} \geq 0$$
$$-x_{97} + x_{98} \geq 0 \qquad -x_{97} + x_{110} \geq 0 \qquad -x_{97} + x_{121} \geq 0$$
$$-x_{97} + x_{132} \geq 0 \qquad -x_{97} + x_{127} \geq 0 \qquad -x_{97} + x_{138} \geq 0$$
$$-x_{97} + x_{105} \geq 0 \qquad -x_{127} + x_{129} \geq 0 \qquad -x_{138} + x_{140} \geq 0$$
$$-x_{105} + x_{107} \geq 0 \qquad -x_{116} + x_{118} \geq 0 \qquad -x_{91} + x_{92} \geq 0$$

$$
\begin{array}{rrr}
-x_{132} + x_{135} \geq 0 & -x_{121} + x_{124} \geq 0 & -x_{110} + x_{113} \geq 0 \\
-x_{98} + x_{101} \geq 0 & -x_{101} + x_{102} \geq 0 & -x_{113} + x_{102} \geq 0 \\
-x_{124} + x_{102} \geq 0 & -x_{135} + x_{102} \geq 0 & -x_{118} + x_{102} \geq 0 \\
-x_{107} + x_{102} \geq 0 & -x_{140} + x_{102} \geq 0 & -x_{129} + x_{102} \geq 0 \\
-x_{102} + x_{143} \geq 0 & -x_{143} + x_{161} \geq 0 & -x_{143} + x_{169} \geq 0 \\
-x_{143} + x_{153} \geq 0 & -x_{143} + x_{144} \geq 0 & -x_{143} + x_{165} \geq 0 \\
-x_{143} + x_{173} \geq 0 & -x_{143} + x_{157} \geq 0 & -x_{143} + x_{149} \geq 0 \\
-x_{165} + x_{167} \geq 0 & -x_{173} + x_{175} \geq 0 & -x_{149} + x_{151} \geq 0 \\
-x_{157} + x_{159} \geq 0 & -x_{92} + x_{94} \geq 0 & -x_{144} + x_{146} \geq 0 \\
-x_{169} + x_{171} \geq 0 & -x_{178} + x_{181} \geq 0 & -x_{153} + x_{155} \geq 0 \\
-x_{161} + x_{163} \geq 0 & -x_{163} + x_{147} \geq 0 & -x_{155} + x_{147} \geq 0 \\
-x_{171} + x_{147} \geq 0 & -x_{146} + x_{147} \geq 0 & -x_{159} + x_{147} \geq 0 \\
-x_{151} + x_{147} \geq 0 & -x_{175} + x_{147} \geq 0 & -x_{167} + x_{147} \geq 0 \\
-x_{147} + x_{95} \geq 0 & -x_{181} + x_{95} \geq 0 & -x_{94} + x_{95} \geq 0 \\
-x_{193} + x_{195} \geq 0 & -x_{195} + 5760 \geq 0 & -x_{95} + 5760 \geq 0 \\
-x_{200} + 5760 \geq 0 & -x_{97} + x_{116} \geq 0 &
\end{array}
$$

Vertical Gap Constraints

$$
\begin{array}{rrr}
x_{199} \geq 0 & -x_{199} + x_{201} \geq 0 & -x_{201} + x_{15} \geq 0 \\
-x_{15} + x_{16} \geq 0 & -x_{16} + x_{19} \geq 0 & -x_{19} + x_{20} \geq 0 \\
-x_{20} + x_{22} \geq 0 & -x_{22} + x_{24} \geq 0 & -x_{24} + x_{25} \geq 0 \\
-x_{25} + x_{27} \geq 0 & -x_{27} + x_{93} \geq 0 & -x_{27} + x_{28} \geq 0 \\
-x_{28} + x_{30} \geq 0 & -x_{93} + x_{96} \geq 0 & -x_{96} + x_{31} \geq 0 \\
-x_{30} + x_{31} \geq 0 & -x_{31} + x_{33} \geq 0 & -x_{31} + x_{99} \geq 0 \\
-x_{99} + x_{145} \geq 0 & -x_{99} + x_{100} \geq 0 & -x_{100} + x_{103} \geq 0 \\
-x_{33} + x_{35} \geq 0 & -x_{145} + x_{148} \geq 0 & -x_{35} + x_{36} \geq 0 \\
-x_{148} + x_{104} \geq 0 & -x_{103} + x_{104} \geq 0 & -x_{104} + x_{106} \geq 0 \\
-x_{104} + x_{150} \geq 0 & -x_{106} + x_{108} \geq 0 & -x_{150} + x_{152} \geq 0 \\
-x_{36} + x_{203} \geq 0 & -x_{152} + x_{109} \geq 0 & -x_{108} + x_{109} \geq 0 \\
-x_{203} + x_{39} \geq 0 & -x_{39} + x_{40} \geq 0 & -x_{109} + x_{111} \geq 0 \\
-x_{111} + x_{112} \geq 0 & -x_{111} + x_{154} \geq 0 & -x_{40} + x_{43} \geq 0 \\
-x_{43} + x_{44} \geq 0 & -x_{112} + x_{114} \geq 0 & -x_{154} + x_{156} \geq 0 \\
-x_{156} + x_{115} \geq 0 & -x_{114} + x_{115} \geq 0 & -x_{115} + x_{158} \geq 0 \\
-x_{115} + x_{117} \geq 0 & -x_{44} + x_{46} \geq 0 & -x_{46} + x_{57} \geq 0
\end{array}
$$

$$
\begin{array}{lll}
-x_{46}+x_{47}\geq 0 & -x_{158}+x_{160}\geq 0 & -x_{117}+x_{119}\geq 0\\
-x_{119}+x_{120}\geq 0 & -x_{160}+x_{120}\geq 0 & -x_{57}+x_{59}\geq 0\\
-x_{47}+x_{49}\geq 0 & -x_{49}+x_{50}\geq 0 & -x_{59}+x_{50}\geq 0\\
-x_{50}+x_{52}\geq 0 & -x_{50}+x_{61}\geq 0 & -x_{120}+x_{122}\geq 0\\
-x_{122}+x_{162}\geq 0 & -x_{122}+x_{123}\geq 0 & -x_{61}+x_{63}\geq 0\\
-x_{52}+x_{54}\geq 0 & -x_{54}+x_{55}\geq 0 & -x_{63}+x_{55}\geq 0\\
-x_{123}+x_{125}\geq 0 & -x_{162}+x_{164}\geq 0 & -x_{164}+x_{126}\geq 0\\
-x_{125}+x_{126}\geq 0 & -x_{126}+x_{128}\geq 0 & -x_{126}+x_{166}\geq 0\\
-x_{55}+x_{65}\geq 0 & -x_{65}+x_{66}\geq 0 & -x_{65}+x_{84}\geq 0\\
-x_{65}+x_{76}\geq 0 & -x_{128}+x_{130}\geq 0 & -x_{166}+x_{168}\geq 0\\
-x_{168}+x_{131}\geq 0 & -x_{130}+x_{131}\geq 0 & -x_{66}+x_{68}\geq 0\\
-x_{76}+x_{78}\geq 0 & -x_{84}+x_{86}\geq 0 & -x_{86}+x_{69}\geq 0\\
-x_{78}+x_{69}\geq 0 & -x_{68}+x_{69}\geq 0 & -x_{69}+x_{88}\geq 0\\
-x_{131}+x_{133}\geq 0 & -x_{69}+x_{80}\geq 0 & -x_{69}+x_{71}\geq 0\\
-x_{133}+x_{134}\geq 0 & -x_{133}+x_{170}\geq 0 & -x_{134}+x_{136}\geq 0\\
-x_{170}+x_{172}\geq 0 & -x_{172}+x_{137}\geq 0 & -x_{136}+x_{137}\geq 0\\
-x_{137}+x_{139}\geq 0 & -x_{137}+x_{174}\geq 0 & -x_{139}+x_{141}\geq 0\\
-x_{174}+x_{176}\geq 0 & -x_{80}+x_{82}\geq 0 & -x_{71}+x_{73}\geq 0\\
-x_{176}+x_{142}\geq 0 & -x_{141}+x_{142}\geq 0 & -x_{142}+x_{180}\geq 0\\
-x_{88}+x_{90}\geq 0 & -x_{90}+x_{74}\geq 0 & -x_{73}+x_{74}\geq 0\\
-x_{82}+x_{74}\geq 0 & -x_{74}+x_{188}\geq 0 & -x_{180}+x_{183}\geq 0\\
-x_{188}+x_{191}\geq 0 & -x_{191}+x_{204}\geq 0 & -x_{183}+x_{204}\geq 0\\
-x_{204}+x_{194}\geq 0 & -x_{194}+x_{197}\geq 0 & -x_{197}+4320\geq 0
\end{array}
$$

Appendix B

Table Layout Samples

All sample tables with desired width set to 800 pt laid out using the simple HTML algorithm (HTML), area approximation algorithm (AA), ICBM layout algorithm (ICBM), iterative column-widening algorithm (ICW), hybrid HTML iterative column widening algorithm (HTML-ICW), and hybrid area approximation iterative column widening algorithm (AA-ICW). For details about the algorithms see section 6.2.

B.1 2n2-linear

These are a strict extension of XHTML's current table notation and are backwards compatible with it. As an example of their use consider the example:	A preferred constraint with a stronger strength will always be satisfied in preference to one of weaker strength. This is similar to how CSS strengths work. However, what should we do if the conflicting constraints have the same strength?

(HTML)

These are a strict extension of XHTML's current table notation and are backwards compatible with it. As an example of their use consider the example:	A preferred constraint with a stronger strength will always be satisfied in preference to one of weaker strength. This is similar to how CSS strengths work. However, what should we do if the conflicting constraints have the same strength?

(AA)

These are a strict extension of XHTML's current table notation and are backwards compatible with it. As an example of their use consider the example:	A preferred constraint with a stronger strength will always be satisfied in preference to one of weaker strength. This is similar to how CSS strengths work. However, what should we do if the conflicting constraints have the same strength?

(ICBM)

These are a strict extension of XHTML's current table notation and are backwards compatible with it. As an example of their use consider the example:	A preferred constraint with a stronger strength will always be satisfied in preference to one of weaker strength. This is similar to how CSS strengths work. However, what should we do if the conflicting constraints have the same strength?

(ICW)

These are a strict extension of XHTML's current table notation and are backwards compatible with it. As an example of their use consider the example:	A preferred constraint with a stronger strength will always be satisfied in preference to one of weaker strength. This is similar to how CSS strengths work. However, what should we do if the conflicting constraints have the same strength?

(HTML-ICW)

These are a strict extension of XHTML's current table notation and are backwards compatible with it. As an example of their use consider the example:	A preferred constraint with a stronger strength will always be satisfied in preference to one of weaker strength. This is similar to how CSS strengths work. However, what should we do if the conflicting constraints have the same strength?

(AA-ICW)

B.2 multipara

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(HTML)

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(AA)

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(ICBM)

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(ICW)

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(HTML-ICW)

Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.	Tables are one of the most powerful and useful design elements in current web document standards such as (X)HTML, CSS and XSL. Indeed because of their power, tables are frequently (mis)used by web designers to finely control page layout, not just to display tabular information. Unlike for tables provided in many document formatting systems, for example LaTeX, authors do not need to precisely specify the width of the table columns, instead the author may allow these to adapt to the viewing context while still preserving the general design intended by the author.

(AA-ICW)

B.3 simple-brick

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(HTML)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(AA)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(ICBM)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(ICW)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(HTML-ICW)

The first meeting for this year will be at 11am Friday week 18th Feb (this will be the new time and day for the meeting) and will be held fortnightly thereafter. The room is the usual one.	short
also short	SVG 1.2 enables a block of text and graphics to be rendered inside a shape while automatically wrapping the objects into lines using the flowRoot element. The idea is to mirror, as far as practical, the existing SVG text elements.

(AA-ICW)

B.4 cs-schedule

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to learn how to write simple programs.			This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Afternoon 1:00-4:00					
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.				

(HTML)

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to learn how to write simple programs.			This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Afternoon 1:00-4:00					
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.				

(AA)

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to learn how to write simple programs.			This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Afternoon 1:00-4:00					
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.				

(ICBM)

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to have a career in the software industry in the future.			
Afternoon 1:00-4:00		This section is for those who already know something about computer science and intend to learn how to write simple programs.			This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.				

(ICW)

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to have a career in the software industry in the future.			
Afternoon 1:00-4:00		This section is for those who already know something about computer science and intend to learn how to write simple programs.			This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.				

(HTML-ICW)

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to have a career in the software industry in the future.			
Afternoon 1:00-4:00		This section is for those who already know something about computer science and intend to learn how to write simple programs.			This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.				

(AA-ICW)

B.5 counterfeit

.	Company A	Company B	Company C	Company D	Company E	Company F	Company G	Company H	Company I
Number of employees	1-10	1	5	1-10	50+	50+	50+	50+	50+
Do you have inventory on hand? If so, how much?	$950,000	Limited inventory on hand	Approximately $200,000	$300,000	$10 million	$15 million	$6 million	Approximately $150 million at list price. 35,000 networking products plus 100,000 accessories on hand.	$40 million
Do you get equipment from other resellers?	Yes, resellers, leasing companies, etc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes, as well as end-user customers and integrators.	Yes
Payment terms?	Net 30	Prepay through net 30	Prepay through net 45	No answer provided	COD, credit card, net 30	Net 30	Prepay, COD, net 1, net 5, net 30, net 60	Prepay, COD, credit card, wire transfer, net 30, net 60, extended terms may be available based on payment history and credit check	Credit card through net 90
Warranty?	90-day warranty	90-day repair or replace	Up to one year replacement	No answer provided	90 days to brokers	30-day replacement	90 days wholesale, one year end user	One-year advanced replacement included with every purchase of preowned gear, extended warranties available	90-day replacement
Tech support?	Yes, phone support	24/7	Phone / IM: regular business hours	Yes, phone support	Yes, 24/7 in some cases by phone	Yes, 24/7 available for cost	Yes, phone, online, e-mail, IM	Yes, 24/5 phone, IM, e-mail; tier-one support free with every purchase; regular business hours of each office location; 24/7 planned by early 2008	Yes, 8 a.m. to 5 p.m. and 24-hour available phone and in person
Product delivery timeframe?	Most shipments are FedEx Standard, FedEx 2 day	Most parts can be at customer site within 48 hours	Overnight to 2 weeks	Most 1-3 day delivery time	We offer any method, but most product is shipped overnight; we also ship all over the world	9-5 overnight	Vast majority available for overnight delivery	Next day to most locations for in-stock items; otherwise 2-3 days	Overnight in most cases, few days of lead time in others
* Price on: 1 Cisco 6504E (WS-C6504E-S32-GE). Includes Supervisor Engine 32 with GE SFP uplinks and four-slot chassis. Typical Cisco list price: $13,000	$10,250	I do not deal in new equipment	$7,000 + GLCs to be specified when available (thin on market)	$8,995	$8,450	$7,145	$8,000	$7,500 (includes fan and SUP, no power)	$7,600
* Price on: Catalyst 6513 13-Slot Chassis, Dual 4000W AC power supplies, high-speed fan tray, Supervisor 720 Fabric. Typical Cisco list price: $65,995	$28,500	$29,200	$29,900	$30,900	$35,000	$26,850	$29,500	$27,500	$28,000
*Prices subject to change based on availability and quantity ordered									

(HTML)

.	Company A	Company B	Company C	Company D	Company E	Company F	Company G	Company H	Company I
Number of employees	1-10	1	5	1-10	50+	50+	50+	50+	50+
Do you have inventory on hand? If so, how much?	$950,000	Limited inventory on hand	Approximately $200,000	$300,000	$10 million	$15 million	$6 million	Approximately $150 million at list price. 35,000 networking products plus 100,000 accessories on hand.	$40 million
Do you get equipment from other resellers?	Yes, resellers, leasing companies, etc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes, as well as end-user customers and integrators.	Yes
Payment terms?	Net 30	Prepay through net 30	Prepay through net 45	No answer provided	COD, credit card, net 30	Net 30	Prepay, COD, net 1, net 5, net 30, net 60	Prepay, COD, credit card, wire transfer, net 30, net 60, extended terms may be available based on payment history and credit check	Credit card through net 90
Warranty?	90-day warranty	90-day repair or replace	Up to one year replacement	No answer provided	90 days to brokers	30-day replacement	90 days wholesale, one year end user	One-year advanced replacement included with every purchase of preowned gear, extended warranties available	90-day replacement
Tech support?	Yes, phone support	24/7	Phone / IM: regular business hours	Yes, phone support	Yes, 24/7 in some cases by phone	Yes, 24/7 available for cost	Yes, phone, online, e-mail, IM	Yes, 24/5 phone, IM, e-mail; tier-one support free with every purchase; regular business hours of each office location; 24/7 planned by early 2008	Yes, 8 a.m. to 5 p.m. and 24-hour available phone and in person
Product delivery timeframe?	Most shipments are FedEx Standard, FedEx 2 day	Most parts can be at customer site within 48 hours	Overnight to 2 weeks	Most 1-3 day delivery time	We offer any method, but most product is shipped overnight; we also ship all over the world	9-5 overnight	Vast majority available for overnight delivery	Next day to most locations for in-stock items; otherwise 2-3 days	Overnight in most cases, few days of lead time in others
* Price on: 1 Cisco 6504E (WS-C6504E-S32-GE). Includes Supervisor Engine 32 with GE SFP uplinks and four-slot chassis. Typical Cisco list price: $13,000	$10,250	I do not deal in new equipment	$7,000 + GLCs to be specified when available (thin on market)	$8,995	$8,450	$7,145	$8,000	$7,500 (includes fan and SUP, no power)	$7,600
* Price on: Catalyst 6513 13-Slot Chassis, Dual 4000W AC power supplies, high-speed fan tray, Supervisor 720 Fabric. Typical Cisco list price: $65,995	$28,500	$29,200	$29,900	$30,900	$35,000	$26,850	$29,500	$27,500	$28,000
*Prices subject to change based on availability and quantity ordered									

(AA)

.	Company A	Company B	Company C	Company D	Company E	Company F	Company G	Company H	Company I
Number of employees	1-10	1	5	1-10	50+	50+	50+	50+	50+
Do you have inventory on hand? If so, how much?	$950,000	Limited inventory on hand	Approximately $200,000	$300,000	$10 million	$15 million	$6 million	Approximately $150 million at list price. 35,000 networking products plus 100,000 accessories on hand.	$40 million
Do you get equipment from other resellers?	Yes, resellers, leasing companies, etc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes, as well as end-user customers and integrators.	Yes
Payment terms?	Net 30	Prepay through net 30	Prepay through net 45	No answer provided	COD, credit card, net 30	Net 30	Prepay, COD, net 1, net 5, net 30, net 60	Prepay, COD, credit card, wire transfer, net 30, net 60, extended terms may be available based on payment history and credit check	Credit card through net 90
Warranty?	90-day warranty	90-day repair or replace	Up to one year replacement	No answer provided	90 days to brokers	30-day replacement	90 days wholesale, one year end user	One-year advanced replacement included with every purchase of preowned gear, extended warranties available	90-day replacement
Tech support?	Yes, phone support	24/7	Phone / IM: regular business hours	Yes, phone support	Yes, 24/7 in some cases by phone	Yes, 24/7 available for cost	Yes, phone, online, e-mail, IM	Yes, 24/5 phone, IM, e-mail; tier-one support free with every purchase; regular business hours of each office location; 24/7 planned by early 2008	Yes, 8 a.m. to 5 p.m. and 24-hour available phone and in person
Product delivery timeframe?	Most shipments are FedEx Standard, FedEx 2 day	Most parts can be at customer site within 48 hours	Overnight to 2 weeks	Most 1-3 day delivery time	We offer any method, but most product is shipped overnight; we also ship all over the world	9-5 overnight	Vast majority available for overnight delivery	Next day to most locations for in-stock items; otherwise 2-3 days	Overnight in most cases, few days of lead time in others
* Price on: 1 Cisco 6504E (WS-C6504E-S32-GE). Includes Supervisor Engine 32 with GE SFP uplinks and four-slot chassis. Typical Cisco list price: $13,000	$10,250	I do not deal in new equipment	$7,000 + GLCs to be specified when available (thin on market)	$8,995	$8,450	$7,145	$8,000	$7,500 (includes fan and SUP, no power)	$7,600
* Price on: Catalyst 6513 13-Slot Chassis, Dual 4000W AC power supplies, high-speed fan tray, Supervisor 720 Fabric. Typical Cisco list price: $65,995	$28,500	$29,200	$29,900	$30,900	$35,000	$26,850	$29,500	$27,500	$28,000
*Prices subject to change based on availability and quantity ordered									

(ICBM)

.	Company A	Company B	Company C	Company D	Company E	Company F	Company G	Company H	Company I
Number of employees	1-10	1	5	1-10	50+	50+	50+	50+	50+
Do you have inventory on hand? If so, how much?	$950,000	Limited inventory on hand	Approximately $200,000	$300,000	$10 million	$15 million	$6 million	Approximately $150 million at list price. 35,000 networking products plus 100,000 accessories on hand.	$40 million
Do you get equipment from other resellers?	Yes, resellers, leasing companies, etc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes, as well as end-user customers and integrators.	Yes
Payment terms?	Net 30	Prepay through net 30	Prepay through net 45	No answer provided	COD, credit card, net 30	Net 30	Prepay, COD, net 1, net 5, net 30, net 60	Prepay, COD, credit card, wire transfer, net 30, net 60, extended terms may be available based on payment history and credit check	Credit card through net 90
Warranty?	90-day warranty	90-day repair or replace	Up to one year replacement	No answer provided	90 days to brokers	30-day replacement	90 days wholesale, one year end user	One-year advanced replacement included with every purchase of preowned gear, extended warranties available	90-day replacement
Tech support?	Yes, phone support	24/7	Phone / IM: regular business hours	Yes, phone support	Yes, 24/7 in some cases by phone	Yes, 24/7 available for cost	Yes, phone, online, e-mail, IM	Yes, 24/5 phone, IM, e-mail; tier-one support free with every purchase; regular business hours of each office location; 24/7 planned by early 2008	Yes, 8 a.m. to 5 p.m. and 24-hour available phone and in person
Product delivery timeframe?	Most shipments are FedEx Standard, FedEx 2 day	Most parts can be at customer site within 48 hours	Overnight to 2 weeks	Most 1-3 day delivery time	We offer any method, but most product is shipped overnight; we also ship all over the world	9-5 overnight	Vast majority available for overnight delivery	Next day to most locations for in-stock items; otherwise 2-3 days	Overnight in most cases, few days of lead time in others
* Price on: 1 Cisco 6504E (WS-C6504E-S32-GE). Includes Supervisor Engine 32 with GE SFP uplinks and four-slot chassis. Typical Cisco list price: $13,000	$10,250	I do not deal in new equipment	$7,000 + GLCs to be specified when available (thin on market)	$8,995	$8,450	$7,145	$8,000	$7,500 (includes fan and SUP, no power)	$7,600
* Price on: Catalyst 6513 13-Slot Chassis, Dual 4000W AC power supplies, high-speed fan tray, Supervisor 720 Fabric. Typical Cisco list price: $65,995	$28,500	$29,200	$29,900	$30,900	$35,000	$26,850	$29,500	$27,500	$28,000
*Prices subject to change based on availability and quantity ordered									

(ICW)

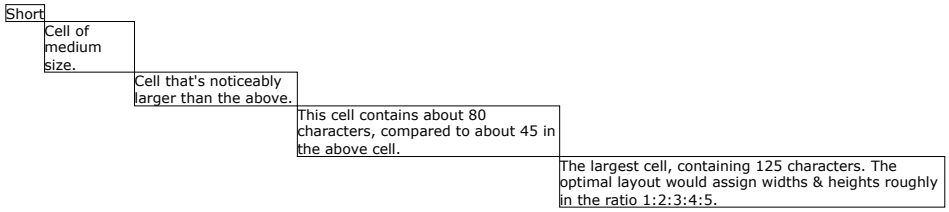
.	Company A	Company B	Company C	Company D	Company E	Company F	Company G	Company H	Company I
Number of employees	1-10	1	5	1-10	50+	50+	50+	50+	50+
Do you have inventory on hand? If so, how much?	$950,000	Limited inventory on hand	Approximately $200,000	$300,000	$10 million	$15 million	$6 million	Approximately $150 million at list price. 35,000 networking products plus 100,000 accessories on hand.	$40 million
Do you get equipment from other resellers?	Yes, resellers, leasing companies, etc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes, as well as end-user customers and integrators.	Yes
Payment terms?	Net 30	Prepay through net 30	Prepay through net 45	No answer provided	COD, credit card, net 30	Net 30	Prepay, COD, net 1, net 5, net 30, net 60	Prepay, COD, credit card, wire transfer, net 30, net 60, extended terms may be available based on payment history and credit check	Credit card through net 90
Warranty?	90-day warranty	90-day repair or replace	Up to one year replacement	No answer provided	90 days to brokers	30-day replacement	90 days wholesale, one year end user	One-year advanced replacement included with every purchase of preowned gear, extended warranties available	90-day replacement
Tech support?	Yes, phone support	24/7	Phone / IM: regular business hours	Yes, phone support	Yes, 24/7 in some cases by phone	Yes, 24/7 available for cost	Yes, phone, online, e-mail, IM	Yes, 24/5 phone, IM, e-mail; tier-one support free with every purchase; regular business hours of each office location; 24/7 planned by early 2008	Yes, 8 a.m. to 5 p.m. and 24-hour available phone and in person
Product delivery timeframe?	Most shipments are FedEx Standard, FedEx 2 day	Most parts can be at customer site within 48 hours	Overnight to 2 weeks	Most 1-3 day delivery time	We offer any method, but most product is shipped overnight; we also ship all over the world	9-5 overnight	Vast majority available for overnight delivery	Next day to most locations for in-stock items; otherwise 2-3 days	Overnight in most cases, few days of lead time in others
* Price on: 1 Cisco 6504E (WS-C6504E-S32-GE). Includes Supervisor Engine 32 with GE SFP uplinks and four-slot chassis. Typical Cisco list price: $13,000	$10,250	I do not deal in new equipment	$7,000 + GLCs to be specified when available (thin on market)	$8,995	$8,450	$7,145	$8,000	$7,500 (includes fan and SUP, no power)	$7,600
* Price on: Catalyst 6513 13-Slot Chassis, Dual 4000W AC power supplies, high-speed fan tray, Supervisor 720 Fabric. Typical Cisco list price: $65,995	$28,500	$29,200	$29,900	$30,900	$35,000	$26,850	$29,500	$27,500	$28,000
*Prices subject to change based on availability and quantity ordered									

(HTML-ICW)

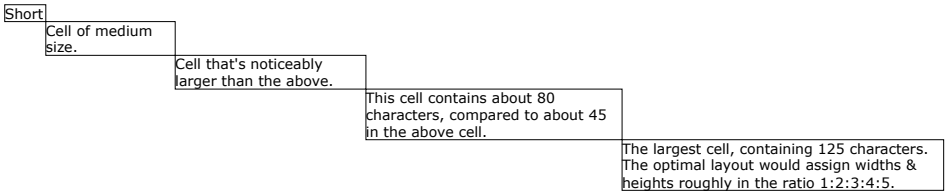
.	Company A	Company B	Company C	Company D	Company E	Company F	Company G	Company H	Company I
Number of employees	1-10	1	5	1-10	50+	50+	50+	50+	50+
Do you have inventory on hand? If so, how much?	$950,000	Limited inventory on hand	Approximately $200,000	$300,000	$10 million	$15 million	$6 million	Approximately $150 million at list price. 35,000 networking products plus 100,000 accessories on hand.	$40 million
Do you get equipment from other resellers?	Yes, resellers, leasing companies, etc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes, as well as end-user customers and integrators.	Yes
Payment terms?	Net 30	Prepay through net 30	Prepay through net 45	No answer provided	COD, credit card, net 30	Net 30	Prepay, COD, net 1, net 5, net 30, net 60	Prepay, COD, credit card, wire transfer, net 30, net 60, extended terms may be available based on payment history and credit check	Credit card through net 90
Warranty?	90-day warranty	90-day repair or replace	Up to one year replacement	No answer provided	90 days to brokers	30-day replacement	90 days wholesale, one year end user	One-year advanced replacement included with every purchase of preowned gear, extended warranties available	90-day replacement
Tech support?	Yes, phone support	24/7	Phone / IM: regular business hours	Yes, phone support	Yes, 24/7 in some cases by phone	Yes, 24/7 available for cost	Yes, phone, online, e-mail, IM	Yes, 24/5 phone, IM, e-mail; tier-one support free with every purchase; regular business hours of each office location; 24/7 planned by early 2008	Yes, 8 a.m. to 5 p.m. and 24-hour available phone and in person
Product delivery timeframe?	Most shipments are FedEx Standard, FedEx 2 day	Most parts can be at customer site within 48 hours	Overnight to 2 weeks	Most 1-3 day delivery time	We offer any method, but most product is shipped overnight; we also ship all over the world	9-5 overnight	Vast majority available for overnight delivery	Next day to most locations for in-stock items; otherwise 2-3 days	Overnight in most cases, few days of lead time in others
* Price on: 1 Cisco 6504E (WS-C6504E-S32-GE). Includes Supervisor Engine 32 with GE SFP uplinks and four-slot chassis. Typical Cisco list price: $13,000	$10,250	I do not deal in new equipment	$7,000 + GLCs to be specified when available (thin on market)	$8,995	$8,450	$7,145	$8,000	$7,500 (includes fan and SUP, no power)	$7,600
* Price on: Catalyst 6513 13-Slot Chassis, Dual 4000W AC power supplies, high-speed fan tray, Supervisor 720 Fabric. Typical Cisco list price: $65,995	$28,500	$29,200	$29,900	$30,900	$35,000	$26,850	$29,500	$27,500	$28,000
*Prices subject to change based on availability and quantity ordered									

(AA-ICW)

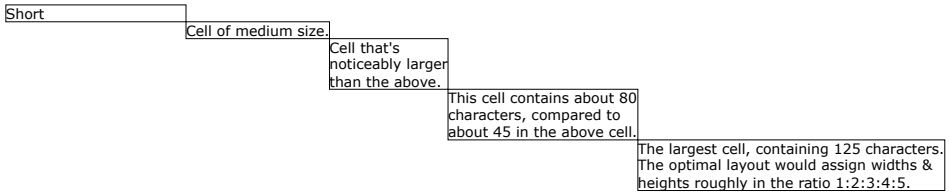
B.6 diagonal5



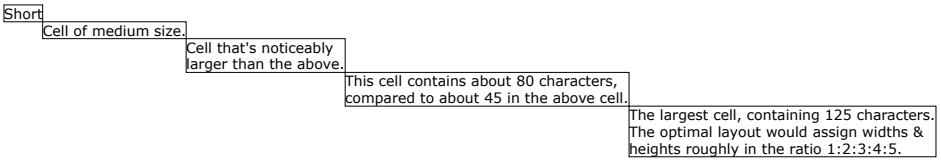
(HTML)



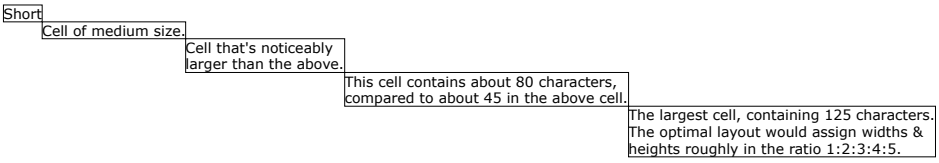
(AA)



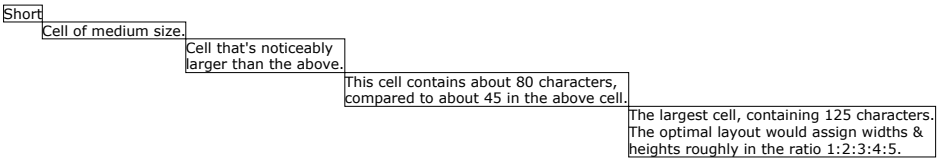
(ICBM)



(ICW)



(HTML-ICW)



(AA-ICW)

B.7 columns

This reminds me of a theory of mine. At the risk of sounding like PFH, I shall inflict it on you: It has long been argued that Korner has the role of "knocking the corners and sharp edges" off young, brilliant, and (almost invariably) arrogant Korner-dwellers. Why are so many of us arrogant? Because when we grow up we are used to being the smartest person in any given room (oops, sorry, my arrogance is showing). So what is the mechanism by which Korner "knocks off" this arrogance? Is everybody in Korner aware that this "knocking off of sharp edges" is a required social role? No! Does everybody in Korner watch out for examples of arrogance and remonstrate against them? No!	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order".	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1].
This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1]. The only way to win this game is to reveal your brilliance while being somewhat modest about it.... and even then some people won't let you get away with it, because they don't have enough social ability to see that you are being modest. By the way, you are not innocent in this field. My memories of Nathan Hurst as a first year is that he was unspeakably bright and insufferably arrogant. Interestingly, you comment that: "Just two days ago I got a phone call from a classmate from 98 who rang up and said "I'm looking for someone smart to work with, do you have Josh's phone number"." I can only suspect that Josh wasn't the choice because he was smarter, he was the choice because he was less arrogant than Nathan Hurst was in 1998, and therefore easier to work with. Your classmate was not aware that people change. Your classmate is wrong of course. I believe I have indicated that I would work with you any day, any time, anywhere. This is because Nathan Hurst today is a very mature and rather complex person. (There is an alternate explanation - your classmate knows this and was hoping that you would ask about the project, get interested, and volunteer.)	1. Long rant follows: I'm sure that you are quite familiar with "changing the rules" in an argument, and I suspect you are quite good at it, though you rarely use it - but here is a topical example: "Dodgy ROI calculations on solar power". What is the correct way to calculate ROI? Easy. I am a Solution Architect. I work on jobs at Telstra and Sensis. My job is to devise the low-cost, quick ROI approaches to Enterprise-level software solutions. These solutions can cost anywhere from $50K to $28,000,000. My boss says "ROI is about a laser-like focus on profit and loss". I'm trusted to produce multi-million dollar decisions for one of Australia's biggest companies - so I should be able to analyse the ROI on Solar Power..... So, what would I do to get the ROI on Solar Power if we suddenly decided to "Go Solar" at Telstra: 1. Design some optimised plant options (this is almost trivial compared to what is about to happen - designing an optimised plant doesn't even deserve it's own number in this numbered list of points). Now gather all costs associated with producing each option of plant. (Get an estimate from the provisioning people, get an estimate of expected life of the product as well, along with an estimate of expected drops in efficiency) 2. Add all the costs associated with supporting and/or maintaining the plant across its lifetime (get an estimate from support - check with Help Desk to see if they are likely to get any calls, talk to Networking to discuss any added costs, etc) 3. Add the cost of disposing of the plant at end-of-life - less any capital that may be returned through selling or recycling of materials. (Get an estimate from Operations, check with provisioning to validate assumptions). 4. Now get an estimate of income for each year of the expected lifespan (go to Operations, validate with Forecasting). 6. The ROI is the point on the graph where ((amortised costs-per-year) + (support costs-per-year)) * (number of years) is less than the total income for that number of years. I am now in a position to compare ROI of Solar Vs ROI of "conventional" energy sources.	BUT WAIT! When I write this up and distribute it "for socialisation" (as the current jargon goes), I get a call from Sales and Marketing. I have forgotten a few things in my ROI. The ROI is wrong. I forgot to consider energy costs of the different options. And I have failed to consider the societal costs associated with pollution. When I point out that these don't have a directly associated cost, Sales and Marketing point out that they have a cost in terms of "Perception" and this has a direct cost in terms of Sales. I asked Operations for their costing estimates, but I failed to ask Sales And Marketing for their costing of the project! Ooops. While I don't consider the "estimates" that Sales and Marketing put forward to be based on any scientifically verifiable principal - here is a hard fact: Without the Sales and Marketing division there would be no sales and no company. If the company is successful, then there must be some veracity to the 'estimates" that Sales and Marketing produce. So I put the estimates in. Then Strategic Directions give me a call. I have failed to consider "Possible synergies associated with alignment with Strategic Imperatives". I don't even know what that means... but I'm pretty beaten down by then, so I put their estimates in. I "Socialise" the new estimates. My boss calls me. It seems that somewhere between the last estimate and this one, I lost my "laser-like focus on profit and loss". :-(So what is my point? Assigning exact numbers to an ROI is like saying that you can provide the exact dimensions for sand. There are an awful lot of sand particles out there. The best you can do is define which sand particle, give the dimensions on that particle - then be ready for them to switch particles on you. When they do the switch, you need to have EXACTLY DEFINED which particle you were talking about. If you didn't, you can't prove that there was a switch. Hmmmmm.... kind of obvious now that I think about it.... sorry, must have been in rant mode. DC.

(HTML)

This reminds me of a theory of mine. At the risk of sounding like PFH, I shall inflict it on you: It has long been argued that Korner has the role of "knocking the corners and sharp edges" off young, brilliant, and (almost invariably) arrogant Korner-dwellers. Why are so many of us arrogant? Because when we grow up we are used to being the smartest person in any given room (oops, sorry, my arrogance is showing). So what is the mechanism by which Korner "knocks off" this arrogance? Is everybody in Korner aware that this "knocking off of sharp edges" is a required social role? No! Does everybody in Korner watch out for examples of arrogance and remonstrate against them? No!	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order".	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1].
This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1]. The only way to win this game is to reveal your brilliance while being somewhat modest about it.... and even then some people won't let you get away with it, because they don't have enough social ability to see that you are being modest. By the way, you are not innocent in this field. My memories of Nathan Hurst as a first year is that he was unspeakably bright and insufferably arrogant. Interestingly, you comment that: "Just two days ago I got a phone call from a classmate from 98 who rang up and said "I'm looking for someone smart to work with, do you have Josh's phone number"." I can only suspect that Josh wasn't the choice because he was smarter, he was the choice because he was less arrogant than Nathan Hurst was in 1998, and therefore easier to work with. Your classmate was not aware that people change. Your classmate is wrong of course. I believe I have indicated that I would work with you any day, any time, anywhere. This is because Nathan Hurst today is a very mature and rather complex person. (There is an alternate explanation - your classmate knows this and was hoping that you would ask about the project, get interested, and volunteer.)	1. Long rant follows: I'm sure that you are quite familiar with "changing the rules" in an argument, and I suspect you are quite good at it, though you rarely use it - but here is a topical example: "Dodgy ROI calculations on solar power". What is the correct way to calculate ROI? Easy. I am a Solution Architect. I work on jobs at Telstra and Sensis. My job is to devise the low-cost, quick ROI approaches to Enterprise-level software solutions. These solutions can cost anywhere from $50K to $28,000,000. My boss says "ROI is about a laser-like focus on profit and loss". I'm trusted to produce multi-million dollar decisions for one of Australia's biggest companies - so I should be able to analyse the ROI on Solar Power..... So, what would I do to get the ROI on Solar Power if we suddenly decided to "Go Solar" at Telstra: 1. Design some optimised plant options (this is almost trivial compared to what is about to happen - designing an optimised plant doesn't even deserve it's own number in this numbered list of points). Now gather all costs associated with producing each option of plant. (Get an estimate from the provisioning people, get an estimate of expected life of the product as well, along with an estimate of expected drops in efficiency) 2. Add all the costs associated with supporting and/or maintaining the plant across its lifetime (get an estimate from support - check with Help Desk to see if they are likely to get any calls, talk to Networking to discuss any added costs, etc) 3. Add the cost of disposing of the plant at end-of-life - less any capital that may be returned through selling or recycling of materials. (Get an estimate from Operations, check with provisioning to validate assumptions). 4. Now get an estimate of income for each year of the expected lifespan (go to Operations, validate with Forecasting). 6. The ROI is the point on the graph where ((amortised costs-per-year) + (support costs-per-year)) * (number of years) is less than the total income for that number of years. I am now in a position to compare ROI of Solar Vs ROI of "conventional" energy sources.	BUT WAIT! When I write this up and distribute it "for socialisation" (as the current jargon goes), I get a call from Sales and Marketing. I have forgotten a few things in my ROI. The ROI is wrong. I forgot to consider energy costs of the different options. And I have failed to consider the societal costs associated with pollution. When I point out that these don't have a directly associated cost, Sales and Marketing point out that they have a cost in terms of "Perception" and this has a direct cost in terms of Sales. I asked Operations for their costing estimates, but I failed to ask Sales And Marketing for their costing of the project! Ooops. While I don't consider the "estimates" that Sales and Marketing put forward to be based on any scientifically verifiable principal - here is a hard fact: Without the Sales and Marketing division there would be no sales and no company. If the company is successful, then there must be some veracity to the 'estimates" that Sales and Marketing produce. So I put the estimates in. Then Strategic Directions give me a call. I have failed to consider "Possible synergies associated with alignment with Strategic Imperatives". I don't even know what that means... but I'm pretty beaten down by then, so I put their estimates in. I "Socialise" the new estimates. My boss calls me. It seems that somewhere between the last estimate and this one, I lost my "laser-like focus on profit and loss". :-(So what is my point? Assigning exact numbers to an ROI is like saying that you can provide the exact dimensions for sand. There are an awful lot of sand particles out there. The best you can do is define which sand particle, give the dimensions on that particle - then be ready for them to switch particles on you. When they do the switch, you need to have EXACTLY DEFINED which particle you were talking about. If you didn't, you can't prove that there was a switch. Hmmmmm.... kind of obvious now that I think about it.... sorry, must have been in rant mode. DC.

(AA)

This reminds me of a theory of mine. At the risk of sounding like PFH, I shall inflict it on you: It has long been argued that Korner has the role of "knocking the corners and sharp edges" off young, brilliant, and (almost invariably) arrogant Korner-dwellers. Why are so many of us arrogant? Because when we grow up we are used to being the smartest person in any given room (oops, sorry, my arrogance is showing). So what is the mechanism by which Korner "knocks off" this arrogance? Is everybody in Korner aware that this "knocking off of sharp edges" is a required social role? No! Does everybody in Korner watch out for examples of arrogance and remonstrate against them? No!	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order".	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1].
This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1]. The only way to win this game is to reveal your brilliance while being somewhat modest about it.... and even then some people won't let you get away with it, because they don't have enough social ability to see that you are being modest. By the way, you are not innocent in this field. My memories of Nathan Hurst as a first year is that he was unspeakably bright and insufferably arrogant. Interestingly, you comment that: "Just two days ago I got a phone call from a classmate from 98 who rang up and said "I'm looking for someone smart to work with, do you have Josh's phone number"." I can only suspect that Josh wasn't the choice because he was smarter, he was the choice because he was less arrogant than Nathan Hurst was in 1998, and therefore easier to work with. Your classmate was not aware that people change. Your classmate is wrong of course. I believe I have indicated that I would work with you any day, any time, anywhere. This is because Nathan Hurst today is a very mature and rather complex person. (There is an alternate explanation - your classmate knows this and was hoping that you would ask about the project, get interested, and volunteer.)	1. Long rant follows: I'm sure that you are quite familiar with "changing the rules" in an argument, and I suspect you are quite good at it, though you rarely use it - but here is a topical example: "Dodgy ROI calculations on solar power". What is the correct way to calculate ROI? Easy. I am a Solution Architect. I work on jobs at Telstra and Sensis. My job is to devise the low-cost, quick ROI approaches to Enterprise-level software solutions. These solutions can cost anywhere from $50K to $28,000,000. My boss says "ROI is about a laser-like focus on profit and loss". I'm trusted to produce multi-million dollar decisions for one of Australia's biggest companies - so I should be able to analyse the ROI on Solar Power..... So, what would I do to get the ROI on Solar Power if we suddenly decided to "Go Solar" at Telstra: 1. Design some optimised plant options (this is almost trivial compared to what is about to happen - designing an optimised plant doesn't even deserve it's own number in this numbered list of points). Now gather all costs associated with producing each option of plant. (Get an estimate from the provisioning people, get an estimate of expected life of the product as well, along with an estimate of expected drops in efficiency) 2. Add all the costs associated with supporting and/or maintaining the plant across its lifetime (get an estimate from support - check with Help Desk to see if they are likely to get any calls, talk to Networking to discuss any added costs, etc) 3. Add the cost of disposing of the plant at end-of-life - less any capital that may be returned through selling or recycling of materials. (Get an estimate from Operations, check with provisioning to validate assumptions). 4. Now get an estimate of income for each year of the expected lifespan (go to Operations, validate with Forecasting). 6. The ROI is the point on the graph where ((amortised costs-per-year) + (support costs-per-year)) * (number of years) is less than the total income for that number of years. I am now in a position to compare ROI of Solar Vs ROI of "conventional" energy sources.	BUT WAIT! When I write this up and distribute it "for socialisation" (as the current jargon goes), I get a call from Sales and Marketing. I have forgotten a few things in my ROI. The ROI is wrong. I forgot to consider energy costs of the different options. And I have failed to consider the societal costs associated with pollution. When I point out that these don't have a directly associated cost, Sales and Marketing point out that they have a cost in terms of "Perception" and this has a direct cost in terms of Sales. I asked Operations for their costing estimates, but I failed to ask Sales And Marketing for their costing of the project! Ooops. While I don't consider the "estimates" that Sales and Marketing put forward to be based on any scientifically verifiable principal - here is a hard fact: Without the Sales and Marketing division there would be no sales and no company. If the company is successful, then there must be some veracity to the 'estimates" that Sales and Marketing produce. So I put the estimates in. Then Strategic Directions give me a call. I have failed to consider "Possible synergies associated with alignment with Strategic Imperatives". I don't even know what that means... but I'm pretty beaten down by then, so I put their estimates in. I "Socialise" the new estimates. My boss calls me. It seems that somewhere between the last estimate and this one, I lost my "laser-like focus on profit and loss". :-(So what is my point? Assigning exact numbers to an ROI is like saying that you can provide the exact dimensions for sand. There are an awful lot of sand particles out there. The best you can do is define which sand particle, give the dimensions on that particle - then be ready for them to switch particles on you. When they do the switch, you need to have EXACTLY DEFINED which particle you were talking about. If you didn't, you can't prove that there was a switch. Hmmmmm.... kind of obvious now that I think about it.... sorry, must have been in rant mode. DC.

(ICBM)

This reminds me of a theory of mine. At the risk of sounding like PFH, I shall inflict it on you: It has long been argued that Korner has the role of "knocking the corners and sharp edges" off young, brilliant, and (almost invariably) arrogant Korner-dwellers. Why are so many of us arrogant? Because when we grow up we are used to being the smartest person in any given room (oops, sorry, my arrogance is showing). So what is the mechanism by which Korner "knocks off" this arrogance? Is everybody in Korner aware that this "knocking off of sharp edges" is a required social role? No! Does everybody in Korner watch out for examples of arrogance and remonstrate against them? No!	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order".	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1].
This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1]. The only way to win this game is to reveal your brilliance while being somewhat modest about it.... and even then some people won't let you get away with it, because they don't have enough social ability to see that you are being modest. By the way, you are not innocent in this field. My memories of Nathan Hurst as a first year is that he was unspeakably bright and insufferably arrogant. Interestingly, you comment that: "Just two days ago I got a phone call from a classmate from 98 who rang up and said "I'm looking for someone smart to work with, do you have Josh's phone number"." I can only suspect that Josh wasn't the choice because he was smarter, he was the choice because he was less arrogant than Nathan Hurst was in 1998, and therefore easier to work with. Your classmate was not aware that people change. Your classmate is wrong of course. I believe I have indicated that I would work with you any day, any time, anywhere. This is because Nathan Hurst today is a very mature and rather complex person. (There is an alternate explanation - your classmate knows this and was hoping that you would ask about the project, get interested, and volunteer.)	1. Long rant follows: I'm sure that you are quite familiar with "changing the rules" in an argument, and I suspect you are quite good at it, though you rarely use it - but here is a topical example: "Dodgy ROI calculations on solar power". What is the correct way to calculate ROI? Easy. I am a Solution Architect. I work on jobs at Telstra and Sensis. My job is to devise the low-cost, quick ROI approaches to Enterprise-level software solutions. These solutions can cost anywhere from $50K to $28,000,000. My boss says "ROI is about a laser-like focus on profit and loss". I'm trusted to produce multi-million dollar decisions for one of Australia's biggest companies - so I should be able to analyse the ROI on Solar Power..... So, what would I do to get the ROI on Solar Power if we suddenly decided to "Go Solar" at Telstra: 1. Design some optimised plant options (this is almost trivial compared to what is about to happen - designing an optimised plant doesn't even deserve it's own number in this numbered list of points). Now gather all costs associated with producing each option of plant. (Get an estimate from the provisioning people, get an estimate of expected life of the product as well, along with an estimate of expected drops in efficiency) 2. Add all the costs associated with supporting and/or maintaining the plant across its lifetime (get an estimate from support - check with Help Desk to see if they are likely to get any calls, talk to Networking to discuss any added costs, etc) 3. Add the cost of disposing of the plant at end-of-life - less any capital that may be returned through selling or recycling of materials. (Get an estimate from Operations, check with provisioning to validate assumptions). 4. Now get an estimate of income for each year of the expected lifespan (go to Operations, validate with Forecasting). 6. The ROI is the point on the graph where ((amortised costs-per-year) + (support costs-per-year)) * (number of years) is less than the total income for that number of years. I am now in a position to compare ROI of Solar Vs ROI of "conventional" energy sources.	BUT WAIT! When I write this up and distribute it "for socialisation" (as the current jargon goes), I get a call from Sales and Marketing. I have forgotten a few things in my ROI. The ROI is wrong. I forgot to consider energy costs of the different options. And I have failed to consider the societal costs associated with pollution. When I point out that these don't have a directly associated cost, Sales and Marketing point out that they have a cost in terms of "Perception" and this has a direct cost in terms of Sales. I asked Operations for their costing estimates, but I failed to ask Sales And Marketing for their costing of the project! Ooops. While I don't consider the "estimates" that Sales and Marketing put forward to be based on any scientifically verifiable principal - here is a hard fact: Without the Sales and Marketing division there would be no sales and no company. If the company is successful, then there must be some veracity to the 'estimates" that Sales and Marketing produce. So I put the estimates in. Then Strategic Directions give me a call. I have failed to consider "Possible synergies associated with alignment with Strategic Imperatives". I don't even know what that means... but I'm pretty beaten down by then, so I put their estimates in. I "Socialise" the new estimates. My boss calls me. It seems that somewhere between the last estimate and this one, I lost my "laser-like focus on profit and loss". :-(So what is my point? Assigning exact numbers to an ROI is like saying that you can provide the exact dimensions for sand. There are an awful lot of sand particles out there. The best you can do is define which sand particle, give the dimensions on that particle - then be ready for them to switch particles on you. When they do the switch, you need to have EXACTLY DEFINED which particle you were talking about. If you didn't, you can't prove that there was a switch. Hmmmmm.... kind of obvious now that I think about it.... sorry, must have been in rant mode. DC.

(ICW)

This reminds me of a theory of mine. At the risk of sounding like PFH, I shall inflict it on you: It has long been argued that Korner has the role of "knocking the corners and sharp edges" off young, brilliant, and (almost invariably) arrogant Korner-dwellers. Why are so many of us arrogant? Because when we grow up we are used to being the smartest person in any given room (oops, sorry, my arrogance is showing). So what is the mechanism by which Korner "knocks off" this arrogance? Is everybody in Korner aware that this "knocking off of sharp edges" is a required social role? No! Does everybody in Korner watch out for examples of arrogance and remonstrate against them? No!	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order".	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1].
This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1]. The only way to win this game is to reveal your brilliance while being somewhat modest about it.... and even then some people won't let you get away with it, because they don't have enough social ability to see that you are being modest. By the way, you are not innocent in this field. My memories of Nathan Hurst as a first year is that he was unspeakably bright and insufferably arrogant. Interestingly, you comment that: "Just two days ago I got a phone call from a classmate from 98 who rang up and said "I'm looking for someone smart to work with, do you have Josh's phone number"." I can only suspect that Josh wasn't the choice because he was smarter, he was the choice because he was less arrogant than Nathan Hurst was in 1998, and therefore easier to work with. Your classmate was not aware that people change. Your classmate is wrong of course. I believe I have indicated that I would work with you any day, any time, anywhere. This is because Nathan Hurst today is a very mature and rather complex person. (There is an alternate explanation - your classmate knows this and was hoping that you would ask about the project, get interested, and volunteer.)	1. Long rant follows: I'm sure that you are quite familiar with "changing the rules" in an argument, and I suspect you are quite good at it, though you rarely use it - but here is a topical example: "Dodgy ROI calculations on solar power". What is the correct way to calculate ROI? Easy. I am a Solution Architect. I work on jobs at Telstra and Sensis. My job is to devise the low-cost, quick ROI approaches to Enterprise-level software solutions. These solutions can cost anywhere from $50K to $28,000,000. My boss says "ROI is about a laser-like focus on profit and loss". I'm trusted to produce multi-million dollar decisions for one of Australia's biggest companies - so I should be able to analyse the ROI on Solar Power..... So, what would I do to get the ROI on Solar Power if we suddenly decided to "Go Solar" at Telstra: 1. Design some optimised plant options (this is almost trivial compared to what is about to happen - designing an optimised plant doesn't even deserve it's own number in this numbered list of points). Now gather all costs associated with producing each option of plant. (Get an estimate from the provisioning people, get an estimate of expected life of the product as well, along with an estimate of expected drops in efficiency) 2. Add all the costs associated with supporting and/or maintaining the plant across its lifetime (get an estimate from support - check with Help Desk to see if they are likely to get any calls, talk to Networking to discuss any added costs, etc) 3. Add the cost of disposing of the plant at end-of-life - less any capital that may be returned through selling or recycling of materials. (Get an estimate from Operations, check with provisioning to validate assumptions). 4. Now get an estimate of income for each year of the expected lifespan (go to Operations, validate with Forecasting). 6. The ROI is the point on the graph where ((amortised costs-per-year) + (support costs-per-year)) * (number of years) is less than the total income for that number of years. I am now in a position to compare ROI of Solar Vs ROI of "conventional" energy sources.	BUT WAIT! When I write this up and distribute it "for socialisation" (as the current jargon goes), I get a call from Sales and Marketing. I have forgotten a few things in my ROI. The ROI is wrong. I forgot to consider energy costs of the different options. And I have failed to consider the societal costs associated with pollution. When I point out that these don't have a directly associated cost, Sales and Marketing point out that they have a cost in terms of "Perception" and this has a direct cost in terms of Sales. I asked Operations for their costing estimates, but I failed to ask Sales And Marketing for their costing of the project! Ooops. While I don't consider the "estimates" that Sales and Marketing put forward to be based on any scientifically verifiable principal - here is a hard fact: Without the Sales and Marketing division there would be no sales and no company. If the company is successful, then there must be some veracity to the 'estimates" that Sales and Marketing produce. So I put the estimates in. Then Strategic Directions give me a call. I have failed to consider "Possible synergies associated with alignment with Strategic Imperatives". I don't even know what that means... but I'm pretty beaten down by then, so I put their estimates in. I "Socialise" the new estimates. My boss calls me. It seems that somewhere between the last estimate and this one, I lost my "laser-like focus on profit and loss". :-(So what is my point? Assigning exact numbers to an ROI is like saying that you can provide the exact dimensions for sand. There are an awful lot of sand particles out there. The best you can do is define which sand particle, give the dimensions on that particle - then be ready for them to switch particles on you. When they do the switch, you need to have EXACTLY DEFINED which particle you were talking about. If you didn't, you can't prove that there was a switch. Hmmmmm.... kind of obvious now that I think about it.... sorry, must have been in rant mode. DC.

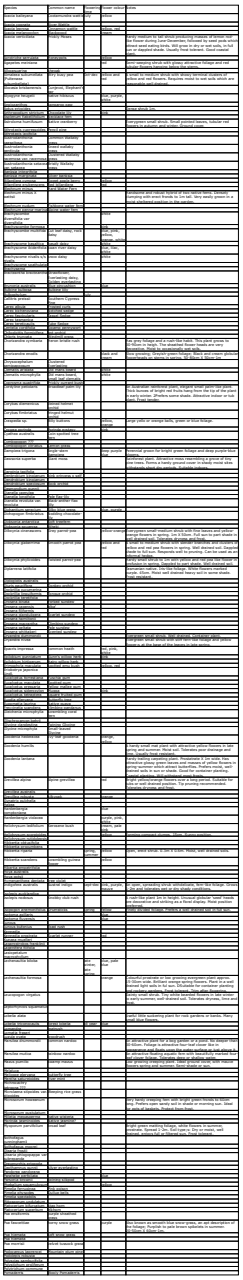
(HTML-ICW)

This reminds me of a theory of mine. At the risk of sounding like PFH, I shall inflict it on you: It has long been argued that Korner has the role of "knocking the corners and sharp edges" off young, brilliant, and (almost invariably) arrogant Korner-dwellers. Why are so many of us arrogant? Because when we grow up we are used to being the smartest person in any given room (oops, sorry, my arrogance is showing). So what is the mechanism by which Korner "knocks off" this arrogance? Is everybody in Korner aware that this "knocking off of sharp edges" is a required social role? No!	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order".	So how does it happen? Simple (and here comes the PFH-like theory). It is a "clash of arrogance". Every time you try to show that you are smart, someone else will try to show that they are smarter, because this supports the belief structure that they grew up with. The resulting trauma causes a simple behavioural aversion to displaying arrogance. This is not always the selfless act of a caring and loving environment (though there may be some elements of this in it....). It is more the establishment of a "social pecking order". This is further complicated by the fact that there are different ways to be smart. Some of us are really bright at maths and logic, but bad at written communication, or poor at reading and understanding social situations, or not great with verbal interactions. Others are fast, witty, verbally quick witted, but not so great in the maths / programming / logic area. This provides endless extra opportunities in the "clash of arrogance" for one person to make another feel stupid. If you demonstrate that something is true, a verbally adroit person will simply "change the rules" to prove that you are wrong [1]. The only way to win this game is to reveal your brilliance while being somewhat modest about it.... and even then some people won't let you get away with it, because they don't have enough social ability to see that you are being modest. By the way, you are not innocent in this field. My memories of Nathan Hurst as a first year is that he was unspeakably bright and insufferably arrogant. Interestingly, you comment that: "Just two days ago I got a phone call from a classmate from 98 who rang up and said "I'm looking for someone smart to work with, do you have Josh's phone number"." I can only suspect that Josh wasn't the choice because he was smarter, he was the choice because he was less arrogant than Nathan Hurst was in 1998, and therefore easier to work with. Your classmate was not aware that people change. Your classmate is wrong of course. I believe I have indicated that I would work with you any day, any time, anywhere. This is because Nathan Hurst today is a very mature and rather complex person. (There is an alternate explanation - your classmate knows this and was hoping that you would ask about the project, get interested, and volunteer.)
Does everybody in Korner watch out for examples of arrogance and remonstrate against them? No!	1. Long rant follows: I'm sure that you are quite familiar with "changing the rules" in an argument, and I suspect you are quite good at it, though you rarely use it - but here is a topical example: "Dodgy ROI calculations on solar power". What is the correct way to calculate ROI? Easy. I am a Solution Architect. I work on jobs at Telstra and Sensis. My job is to devise the low-cost, quick ROI approaches to Enterprise-level software solutions. These solutions can cost anywhere from $50K to $28,000,000. My boss says "ROI is about a laser-like focus on profit and loss". I'm trusted to produce multi-million dollar decisions for one of Australia's biggest companies - so I should be able to analyse the ROI on Solar Power..... So, what would I do to get the ROI on Solar Power if we suddenly decided to "Go Solar" at Telstra: 1. Design some optimised plant options (this is almost trivial compared to what is about to happen - designing an optimised plant doesn't even deserve it's own number in this numbered list of points). Now gather all costs associated with producing each option of plant. (Get an estimate from the provisioning people, get an estimate of expected life of the product as well, along with an estimate of expected drops in efficiency) 2. Add all the costs associated with supporting and/or maintaining the plant across its lifetime (get an estimate from support - check with Help Desk to see if they are likely to get any calls, talk to Networking to discuss any added costs, etc) 3. Add the cost of disposing of the plant at end-of-life - less any capital that may be returned through selling or recycling of materials. (Get an estimate from Operations, check with provisioning to validate assumptions). 4. Now get an estimate of income for each year of the expected lifespan (go to Operations, validate with Forecasting). 6. The ROI is the point on the graph where ((amortised costs-per-year) + (support costs-per-year)) * (number of years) is less than the total income for that number of years. I am now in a position to compare ROI of Solar Vs ROI of "conventional" energy sources.	BUT WAIT! When I write this up and distribute it "for socialisation" (as the current jargon goes), I get a call from Sales and Marketing. I have forgotten a few things in my ROI. The ROI is wrong. I forgot to consider energy costs of the different options. And I have failed to consider the societal costs associated with pollution. When I point out that these don't have a directly associated cost, Sales and Marketing point out that they have a cost in terms of "Perception" and this has a direct cost in terms of Sales. I asked Operations for their costing estimates, but I failed to ask Sales And Marketing for their costing of the project! Ooops. While I don't consider the "estimates" that Sales and Marketing put forward to be based on any scientifically verifiable principal - here is a hard fact: Without the Sales and Marketing division there would be no sales and no company. If the company is successful, then there must be some veracity to the 'estimates" that Sales and Marketing produce. So I put the estimates in. Then Strategic Directions give me a call. I have failed to consider "Possible synergies associated with alignment with Strategic Imperatives". I don't even know what that means... but I'm pretty beaten down by then, so I put their estimates in. I "Socialise" the new estimates. My boss calls me. It seems that somewhere between the last estimate and this one, I lost my "laser-like focus on profit and loss". :-(So what is my point? Assigning exact numbers to an ROI is like saying that you can provide the exact dimensions for sand. There are an awful lot of sand particles out there. The best you can do is define which sand particle, give the dimensions on that particle - then be ready for them to switch particles on you. When they do the switch, you need to have EXACTLY DEFINED which particle you were talking about. If you didn't, you can't prove that there was a switch. Hmmmmm.... kind of obvious now that I think about it.... sorry, must have been in rant mode. DC.

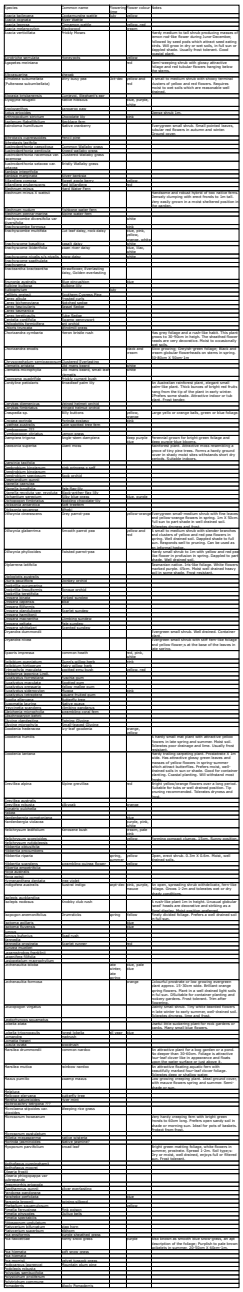
(AA-ICW)

B.8 plants200

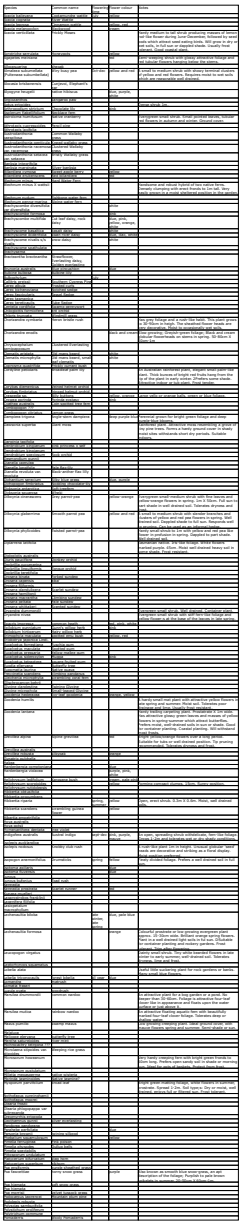
(HTML)



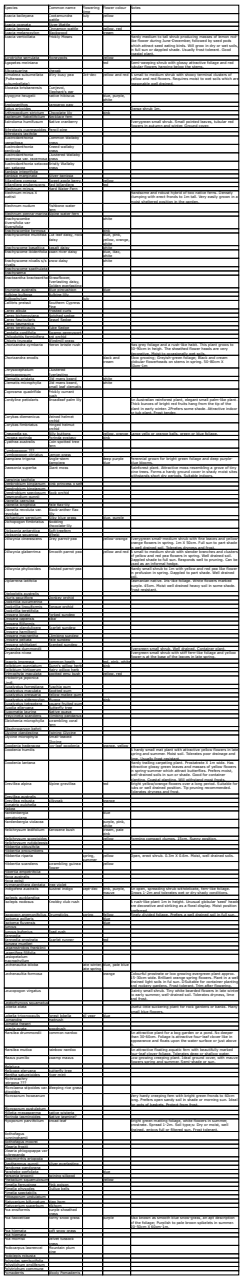
(AA)



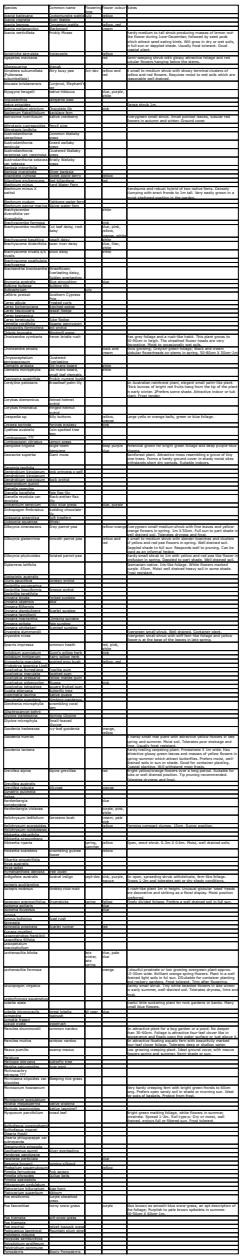
(ICBM)



(ICW)



(HTML-ICW)



(AA-ICW)

Appendix C

Evaluation Raw Data

The following table lists the raw results of the evaluation study of chapter 6. Column *Algorithm* contains the name of the table layout algorithm. The ICBM algorithm is listed twice, on the first run, each table width was set to 450, 500, ..., 1200 pt an on the second run the table widths are set to the best width achieved by the ICW algorithm. Column *Sample* names the table layout sample. Column t contains the time in milliseconds including text measurements, column t_{ex} is the time in milliseconds excluding text measurements, i.e., only including the layout algorithm itself. Column $w_{\max}$ contains the maximum permitted table width, column w the achieved width and column h the achieved height.

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
ICW						
	cs-schedule					
		78	0	450	450	258.75
		72	0	500	491	201.25
		73	1	550	491	201.25
		73	0	600	558	187
		73	0	650	626.75	172.5
		73	1	700	664	158
		72	1	750	709.75	143.75
		73	1	800	785.25	129
		72	1	850	785.25	129
		73	1	900	785.25	129
		73	1	950	939	115
		73	1	1000	939	115
		74	1	1050	939	115
		75	1	1100	939	115

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		75	1	1150	939	115
		74	1	1200	1177.75	101
	multipara					
		98	2	450	429.25	258.75
		86	2	500	472	230
		86	2	550	549	201.25
		86	2	600	549	201.25
		86	3	650	630	172.5
		86	2	700	699.25	158
		86	2	750	699.25	158
		86	2	800	774.75	143.75
		87	2	850	774.75	143.75
		86	3	900	851	129
		90	3	950	925.75	115
		92	3	1000	925.75	115
		88	3	1050	925.75	115
		88	3	1100	925.75	115
		88	3	1150	1102.75	101
		88	3	1200	1102.75	101
	plants200					
		1642	491	450	450	8653.75
		2016	876	500	499	7303
		2063	918	550	549	6296
		2180	1038	600	599	5506
		2298	1153	650	650	5016.88
		2345	1198	700	699.125	4628.75
		2439	1293	750	748.625	4398.75
		2435	1287	800	799.875	4140
		2470	1325	850	846.75	3982
		2565	1415	900	895.125	3853
		2564	1415	950	949.25	3723
		2577	1427	1000	991.875	3623
		2614	1464	1050	1044.25	3551
		2624	1473	1100	1092.88	3479
		2659	1505	1150	1143.63	3421
		2656	1506	1200	1194.88	3393
	2n2-linear					
		31	0	450	442.625	86.25
		31	0	500	494.875	71.875
		30	0	550	494.875	71.875

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		30	0	600	495	71.875
		30	0	650	613.25	57.5
		30	0	700	613.25	57.5
		30	0	750	613	57.5
		30	0	800	613.25	57.5
		30	0	850	819.25	43.125
		30	0	900	819.25	43
		30	0	950	819.25	43
		30	0	1000	819.25	43
		30	0	1050	819.25	43.125
		30	0	1100	819.25	43
		30	0	1150	819.25	43
		31	0	1200	819.25	43
	simple-brick					
		47	0	450	418.25	100.625
		38	0	500	418.25	101
		38	0	550	511.25	86.25
		38	0	600	511	86
		38	0	650	602	71.875
		38	0	700	602	71.875
		38	0	750	739.5	57.5
		38	0	800	740	58
		38	0	850	740	58
		38	0	900	740	57.5
		39	0	950	739.5	58
		38	0	1000	739.5	58
		38	0	1050	739.5	58
		38	0	1100	739.5	58
		38	0	1150	739.5	57.5
		38	0	1200	1159.13	43.125
	columns					
		697	49	450	449.25	1868.75
		687	54	500	494.875	1653.13
		698	58	550	549	1480.63
		697	60	600	599.125	1351.25
		703	64	650	649.25	1236.25
		709	64	700	697.75	1150
		710	66	750	747	1063.75
		749	68	800	797	991.875
		711	68	850	839	934

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		712	69	900	886	877
		715	70	950	941	833.75
		715	71	1000	989.125	790.625
		716	72	1050	1048.25	748
		715	72	1100	1095.25	704
		718	73	1150	1128.88	690
		720	74	1200	1171.88	661.25
	counterfeit					
		247	0	450	779	1207.5
		243	0	500	779.25	1207.5
		242	0	550	779	1207.5
		243	0	600	779	1208
		244	0	650	779	1208
		249	0	700	779	1208
		246	0	750	779.25	1207.5
		247	2	800	796	1078
		250	4	850	848	877
		252	6	900	898	791
		255	7	950	935	747.5
		260	9	1000	987	690
		258	11	1050	1043.25	618
		260	12	1100	1086	589
		258	11	1150	1139.75	561
		258	12	1200	1169.88	546
	diagonal5					
		42	0	450	427.375	259
		29	0	500	493	230
		29	0	550	529.625	201
		29	0	600	564.125	187
		29	1	650	642.375	158
		29	1	700	642.375	158
		29	1	750	714	144
		29	1	800	794.125	129
		29	1	850	794.125	129
		29	1	900	794.125	129
		30	1	950	911.125	115
		29	1	1000	911.125	115
		29	1	1050	1041.13	101
		29	1	1100	1041.13	101
		30	1	1150	1041.13	101

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		30	1	1200	1041.13	101
HTML						
	cs-schedule					
		74	0	450	450	302
		72	0	500	499.875	244.375
		72	0	550	550	201.25
		71	0	600	600.125	186.875
		71	0	650	650	186.875
		71	0	700	700.125	158.125
		72	0	750	750	158
		72	0	800	800	143.75
		72	0	850	849.875	143.75
		75	0	900	899.875	143.75
		76	0	950	950	143.75
		72	0	1000	1000	143.75
		72	0	1050	1050	144
		72	0	1100	1100.13	144
		73	0	1150	1150	143.75
		73	0	1200	1200	143.75
	multipara					
		85	0	450	450	331
		84	0	500	500	316.25
		83	0	550	550	288
		83	0	600	600	259
		83	0	650	650	230
		84	0	700	700	230
		83	0	750	750	215.625
		83	0	800	800	201.25
		83	0	850	850	201
		83	0	900	900	173
		83	0	950	950	173
		83	0	1000	1000	173
		84	0	1050	1050	158
		86	0	1100	1100	158
		85	0	1150	1150	144
		86	0	1200	1200	144
	plants200					
		1143	1	450	450	8811.88
		1147	1	500	500.125	7503.75
		1140	1	550	550	6684.38

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		1148	1	600	600	6195.63
		1183	1	650	650	5706.88
		1148	1	700	700	5419.38
		1145	1	750	750	4988.13
		1145	1	800	800	4772.5
		1145	1	850	850.125	4485
		1147	1	900	900	4226.25
		1146	1	950	950	4011
		1149	1	1000	1000	3809.38
		1148	1	1050	1050	3708.75
		1146	1	1100	1100	3622.5
		1147	1	1150	1150	3565
		1152	1	1200	1199.88	3464
	2n2-linear					
		31	0	450	450	100.625
		30	0	500	500	86
		29	0	550	550	72
		30	0	600	600	72
		30	0	650	650	58
		43	0	700	700	58
		29	0	750	750	58
		29	0	800	800	58
		29	0	850	850	43
		30	0	900	900	43.125
		29	0	950	950	43.125
		29	0	1000	1000	43.125
		30	0	1050	1050	43.125
		30	0	1100	1100	43.125
		30	0	1150	1150	43.125
		30	0	1200	1200	43.125
	simple-brick					
		39	0	450	450	115
		38	0	500	500.125	115
		38	0	550	549.875	86.25
		38	0	600	600	86.25
		38	0	650	650	86.25
		38	0	700	700	86.25
		37	0	750	750	71.875
		37	0	800	800	57.5
		38	0	850	849.875	57.5

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		37	0	900	900	57.5
		38	0	950	950	57.5
		38	0	1000	1000	57.5
		38	0	1050	1050	57.5
		37	0	1100	1100.13	57.5
		38	0	1150	1150	57.5
		38	0	1200	1200	58
	columns					
		638	0	450	450	2098.75
		641	0	500	500	1898
		643	0	550	550	1668
		638	0	600	600	1523.75
		644	0	650	650	1380
		641	0	700	700	1294
		640	0	750	749.875	1208
		638	0	800	799.875	1121.25
		671	0	850	850	1049.38
		648	0	900	899.875	1006.25
		644	0	950	950	949
		645	0	1000	1000	891
		645	0	1050	1049.88	834
		644	0	1100	1100	805
		645	0	1150	1150	776
		645	0	1200	1200	733
	counterfeit					
		246	0	450	779.25	1208
		242	0	500	779.25	1208
		241	0	550	779.25	1208
		242	0	600	779.25	1208
		244	0	650	779.25	1208
		253	0	700	779.25	1208
		246	0	750	779.25	1208
		247	0	800	799.875	1179
		247	0	850	850.125	1078
		250	0	900	900.25	1006
		246	0	950	949.875	891
		247	0	1000	1000.13	848
		248	0	1050	1050	805
		252	0	1100	1100	762
		246	0	1150	1150.25	690

Algorithm	Sample	t	t_{ex}	w_{max}	w	h
		248	0	1200	1199.88	647
	diagonal5					
		29	0	450	450.125	287.5
		28	0	500	499.875	244.375
		28	0	550	550	216
		28	0	600	600	201
		29	0	650	650	201
		28	0	700	700	187
		28	0	750	750	186.875
		28	0	800	800	173
		28	0	850	850	173
		28	0	900	900	143.75
		28	0	950	950	129
		28	0	1000	1.000	129
		28	0	1050	1050	129.375
		28	0	1100	1100	129.375
		28	0	1150	1150.13	129.375
		29	0	1200	1200	129
HTML-ICW						
	cs-schedule					
		75	0	450	450	258.75
		73	0	500	491	201
		72	0	550	491	201
		71	0	600	558	187
		72	0	650	626.75	172.5
		72	0	700	650.375	158.125
		72	0	750	709.75	143.75
		72	0	800	785.25	129.375
		72	0	850	785.25	129
		72	0	900	785.25	129
		72	0	950	939.375	115
		72	0	1000	939.375	115
		72	0	1050	939.375	115
		72	0	1100	939.375	115
		73	0	1150	939.375	115
		73	0	1200	1177.75	101
	multipara					
		86	0	450	429.25	258.75
		84	0	500	472.125	230

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		83	0	550	549	201.25
		83	0	600	549	201.25
		84	0	650	630.375	172.5
		84	0	700	699.25	158.125
		83	0	750	699.25	158.125
		84	0	800	775	143.75
		85	0	850	774.75	144
		85	0	900	851	129
		90	0	950	925.75	115
		84	0	1000	925.75	115
		84	0	1050	925.75	115
		85	0	1100	925.75	115
		85	0	1150	1102.75	101
		85	0	1200	1102.75	100.625
	plants200					
		1445	308	450	449.75	8682.5
		1481	332	500	499.625	7303
		1237	99	550	550	6325
		1395	230	600	599	5850.63
		1274	129	650	650	5505.63
		1381	193	700	700	5175
		1223	69	750	750	4773
		1282	136	800	799	4327
		1294	146	850	848	4126
		1301	153	900	898.75	3967.5
		1264	116	950	949.875	3852.5
		1244	97	1000	995.875	3651.25
		1238	91	1050	1048.75	3550.63
		1223	75	1100	1100	3521.88
		1204	54	1150	1149.63	3464.38
		1187	38	1200	1194.25	3406.88
	2n2-linear					
		31	0	450	442.625	86.25
		30	0	500	495	71.875
		30	0	550	495	71.875
		31	0	600	495	71.875
		30	0	650	613.25	57.5
		30	0	700	613.25	57.5
		30	0	750	613.25	57.5
		29	0	800	613.25	57.5

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		30	0	850	819.25	43.125
		30	0	900	819.25	43.125
		29	0	950	819	43.125
		30	0	1000	819	43.125
		30	0	1050	819.25	43.125
		29	0	1100	819.25	43.125
		30	0	1150	819.25	43.125
		30	0	1200	819.25	43.125
	simple-brick					
		39	0	450	418.25	100.625
		38	0	500	418.25	100.625
		38	0	550	511.25	86.25
		38	0	600	511.25	86.25
		38	0	650	602	71.875
		37	0	700	602	71.875
		37	0	750	739.5	57.5
		38	0	800	740	57.5
		38	0	850	740	58
		37	0	900	739.5	58
		38	0	950	740	58
		38	0	1000	740	57.5
		38	0	1050	739.5	58
		37	0	1100	739.5	58
		37	0	1150	739.5	58
		38	0	1200	1159.13	43.125
	columns					
		645	7	450	449.5	2070
		640	7	500	498.5	1797
		653	5	550	547.25	1639
		644	5	600	599.75	1466
		644	5	650	648.75	1265
		652	5	700	699	1208
		648	4	750	749.875	1121
		645	3	800	797.25	1049
		677	3	850	848.75	992
		644	2	900	899.625	978
		645	4	950	942.625	834
		646	2	1000	997.25	834
		646	2	1050	1049	762
		645	2	1100	1092.25	748

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		646	2	1150	1148.13	704
		646	2	1200	1176.88	661
	counterfeit					
		260	0	450	779.25	1207.5
		242	0	500	779.25	1207.5
		243	0	550	779.25	1207.5
		242	0	600	779.25	1207.5
		243	0	650	779.25	1208
		248	0	700	779	1208
		248	0	750	779.25	1208
		247	2	800	795.625	1078
		249	3	850	848	877
		248	3	900	899	791
		249	2	950	937.125	733.125
		249	3	1000	998.875	661.25
		250	4	1050	1043.25	618.125
		252	4	1100	1086	589.375
		249	3	1150	1139.75	560.625
		250	2	1200	1169.88	546.25
	diagonal5					
		30	0	450	427.375	259
		28	0	500	495.5	215.625
		28	0	550	530	201.25
		28	0	600	566	186.875
		29	0	650	642	158.125
		28	0	700	672	158.125
		28	0	750	714	143.75
		28	0	800	794	129.375
		28	0	850	794	129.375
		28	0	900	794	129.375
		28	0	950	911	115
		28	0	1000	911	115
		28	0	1050	1.041	100.625
		28	0	1100	1.041	100.625
		29	0	1150	1.041	100.625
		29	0	1200	1.041	100.625
AA						
	cs-schedule					
		88	13	450	450	273.125
		85	13	500	500	230

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		84	12	550	550	201.25
		84	13	600	600.125	186.875
		85	13	650	650	172.5
		85	13	700	699.875	158.125
		85	12	750	750	143.75
		85	13	800	799.875	143.75
		85	13	850	850	143.75
		85	13	900	900	143.75
		85	13	950	950	129.375
		85	13	1000	999.875	129
		85	12	1050	1050.25	129
		85	12	1100	1100.13	115
		86	12	1150	1150	115
		87	13	1200	1200	115
	multipara					
		97	11	450	450	258.75
		96	11	500	500	230
		94	11	550	550	215.625
		94	11	600	600	201.25
		95	11	650	650	172.5
		95	11	700	700	172.5
		95	11	750	750	158.125
		94	11	800	800	158
		95	11	850	850	144
		95	11	900	900	144
		94	11	950	950	115
		95	11	1000	1000	115
		95	11	1050	1050	115
		96	11	1100	1100	115
		96	11	1150	1150	115
		96	11	1200	1200	101
	plants200					
		1228	91	450	450	8740
		1242	104	500	500	7417.5
		1240	96	550	550.125	6526.25
		1235	96	600	600	5865
		1267	96	650	649.875	5376.25
		1252	105	700	699.875	4930.63
		1247	100	750	750	4556.88
		1251	105	800	800	4326.88

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		1254	102	850	850	4140
		1255	105	900	900	4010.63
		1257	109	950	950	3838.13
		1264	115	1000	1000	3751.88
		1264	112	1050	1050.13	3665.63
		1257	105	1100	1100	3636.88
		1257	105	1150	1150	3550.63
		1252	100	1200	1200	3493.13
	2n2-linear					
		43	11	450	450	86
		41	11	500	500	86
		41	11	550	550	71.875
		41	11	600	600	71.875
		42	11	650	650	58
		41	11	700	700	58
		42	11	750	750	57.5
		41	11	800	800	57.5
		41	11	850	850	43.125
		42	11	900	900	43.125
		41	11	950	950	43.125
		41	11	1000	1000	43.125
		41	11	1050	1050	43.125
		41	11	1100	1100	43.125
		41	10	1150	1150	43.125
		41	11	1200	1200	43
	simple-brick					
		52	12	450	450	101
		49	11	500	500	100.625
		49	11	550	550	86.25
		50	11	600	600	86.25
		49	11	650	650	72
		49	11	700	700	71.875
		49	11	750	750	57.5
		50	11	800	800	58
		50	11	850	850	58
		49	11	900	900	58
		49	11	950	950	58
		49	11	1000	1000	58
		49	11	1050	1050	58
		49	11	1100	1100	57.5

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		49	11	1150	1150	57.5
		49	11	1200	1200.13	43.125
	columns					
		651	12	450	450	1883.13
		647	12	500	500	1667.5
		653	12	550	550	1509
		651	12	600	600	1366
		651	12	650	650	1279
		650	12	700	700	1179
		652	12	750	750	1078.13
		653	12	800	800	992
		680	12	850	850	934
		658	12	900	900.125	876.875
		657	12	950	950	834
		655	12	1000	1.000	805
		656	12	1050	1050	761.875
		657	12	1100	1100	718.75
		658	12	1150	1150	690
		659	12	1200	1200	661
	counterfeit					
		265	16	450	779.25	1207.5
		259	16	500	779.25	1207.5
		259	16	550	779.25	1207.5
		260	16	600	779.25	1207.5
		262	16	650	779.25	1207.5
		264	16	700	779.25	1207.5
		263	16	750	779.25	1207.5
		267	22	800	800	1078.13
		269	21	850	850	905.625
		268	22	900	900.125	833.75
		270	23	950	950	761.875
		270	22	1000	1000	718.75
		271	24	1050	1050	675.625
		271	22	1100	1100	632.5
		269	22	1150	1150	603.75
		269	22	1200	1200	589.375
	diagonal5					
		43	12	450	450.125	301.875
		41	12	500	500	258.75
		41	12	550	550	230

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		41	12	600	600	201.25
		41	12	650	650	186.875
		40	12	700	700	173
		41	12	750	750	172.5
		40	12	800	800	158
		41	12	850	850	158.125
		41	12	900	899.875	129
		41	12	950	950	129
		41	12	1000	1000.13	129
		41	12	1050	1.050	129.375
		40	12	1100	1.100	115
		41	12	1150	1.150	115
		40	12	1200	1.200	115
AA-ICW						
	complexity					
		56	15	450	404.25	143.75
		53	15	500	465	115
		53	15	550	465.25	115
		54	14	600	465	115
		52	14	650	465.25	115
		53	14	700	465.25	115
		52	14	750	465	115
		53	15	800	465.25	115
		54	15	850	465.25	115
		53	14	900	465.25	115
		53	14	950	465.25	115
		53	14	1000	465.25	115
		53	14	1050	465.25	115
		53	14	1100	465.25	115
		53	14	1150	465.25	115
		53	14	1200	465.25	115
	cs-schedule					
		88	13	450	449.625	258.75
		86	13	500	491.125	201
		85	13	550	491	201.25
		84	13	600	558	186.875
		85	13	650	627	172.5
		86	13	700	650	158
		85	13	750	710	144
		86	13	800	785	129

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		85	13	850	785	129
		85	13	900	785.25	129.375
		86	13	950	939.375	115
		86	13	1000	939.375	115
		85	13	1050	939.375	115
		85	13	1100	939.375	115
		87	13	1150	939.375	115
		86	13	1200	1177.75	100.625
	multipara					
		98	11	450	429.25	258.75
		96	11	500	472	230
		95	11	550	549	201.25
		95	11	600	549	201.25
		96	11	650	630	172.5
		95	11	700	699	158.125
		95	11	750	699.25	158.125
		96	11	800	775	143.75
		95	11	850	774.75	143.75
		95	11	900	851	129.375
		96	11	950	926	115
		95	11	1000	926	115
		96	11	1050	925.75	115
		96	11	1100	925.75	115
		97	11	1150	1102.75	100.625
		97	11	1200	1102.75	100.625
	plants200					
		1369	216	450	449.625	8697
		1361	218	500	499.625	7374
		1393	246	550	550	6483.13
		1407	223	600	599.625	5807.5
		1317	170	650	649.25	5319
		1312	164	700	700	4902
		1325	177	750	749.25	4499.38
		1347	197	800	799.75	4255
		1272	121	850	849	4140
		1348	198	900	894.25	3910
		1315	161	950	946.875	3766.25
		1321	164	1000	995.375	3694.38
		1325	159	1050	1048.63	3636.88
		1333	162	1100	1100	3521.88

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		1325	166	1150	1148.63	3493.13
		1297	136	1200	1194.25	3407
	2n2-linear					
		44	11	450	443	86.25
		41	11	500	494.875	72
		41	11	550	494.875	71.875
		41	11	600	495	72
		41	11	650	613	57.5
		41	11	700	613.25	57.5
		41	11	750	613.25	57.5
		41	11	800	613	58
		41	11	850	819	43
		42	11	900	819	43.125
		41	11	950	819.25	43
		41	11	1000	819.25	43
		41	11	1050	819.25	43
		43	13	1100	819.25	43
		41	11	1150	819.25	43.125
		41	11	1200	819.25	43.125
	simple-brick					
		52	11	450	418.25	101
		50	11	500	418.25	101
		49	11	550	511.25	86
		51	11	600	511.25	86
		49	11	650	602	72
		49	11	700	602	72
		49	11	750	739.5	58
		50	11	800	739.5	58
		49	11	850	739.5	58
		49	11	900	739.5	58
		49	11	950	739.5	58
		49	11	1000	739.5	58
		49	11	1050	739.5	58
		49	11	1100	739.5	58
		49	11	1150	739.5	58
		49	11	1200	1159.13	43
	columns					
		658	18	450	444.5	1883
		652	17	500	499.125	1653.13
		660	17	550	549	1480.63

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		652	15	600	593.75	1366
		663	16	650	649.25	1236
		659	15	700	697.75	1150
		657	14	750	747	1064
		661	14	800	796.5	992
		682	14	850	838.5	934
		658	13	900	885.625	877
		658	14	950	929.75	834
		658	14	1000	994.375	791
		656	13	1050	1043.38	748
		658	13	1100	1095.25	704
		659	13	1150	1133.25	690
		657	13	1200	1171.88	661
	counterfeit					
		264	17	450	779.25	1208
		259	17	500	779.25	1207.5
		258	16	550	779	1207.5
		261	17	600	779	1208
		260	16	650	779	1208
		264	16	700	779	1207.5
		263	16	750	779.25	1208
		268	22	800	796	1078
		270	22	850	848.125	877
		270	24	900	898.5	790.625
		271	24	950	950	733.125
		272	25	1000	998.875	661.25
		272	26	1050	1.049	618
		271	24	1100	1.086	589
		271	24	1150	1.140	561
		272	24	1200	1.170	546
	diagonal5					
		43	13	450	427.375	259
		41	12	500	495.5	215.625
		41	12	550	529.625	201.25
		41	12	600	564	187
		42	12	650	642.375	158.125
		41	12	700	642.375	158.125
		41	12	750	714	144
		40	12	800	794.125	129
		41	12	850	794.125	129

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		41	12	900	794.125	129.375
		41	12	950	911.125	115
		41	12	1000	911.125	115
		41	12	1050	1041.13	101
		40	12	1100	1041.13	101
		41	12	1150	1041.13	101
		41	12	1200	1041.13	101
ICBM						
	complexity					
		48	19	450	450	129.375
		52	20	500	500	115
		51	18	550	550	115
		57	24	600	600	115
		58	25	650	650	115
		56	23	700	700	115
		55	23	750	750	115
		55	23	800	800	115
		55	22	850	850	115
		54	22	900	900	115
		57	23	950	950	115
		56	23	1000	1000	115
		57	24	1050	1050	115
		55	22	1100	1100	115
		55	22	1150	1150	115
		55	23	1200	1200	115
	cs-schedule					
		54	15	450	450	273.125
		60	15	500	500	230
		59	15	550	550	201.25
		59	16	600	600	201.25
		65	15	650	650	186.875
		68	15	700	700	186.875
		67	14	750	750	172.5
		67	14	800	800	143.75
		68	15	850	850	143.75
		67	14	900	900	143.75
		68	15	950	950	129.375
		68	15	1000	1.000	129.375
		73	19	1050	1050	129.375
		73	19	1100	1100	129.375

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		68	14	1150	1150	115
		94	19	1200	1200	115
	multipara					
		55	10	450	450	258.75
		57	10	500	500	230
		57	10	550	550	215.625
		58	10	600	600	201.25
		63	10	650	650	173
		66	10	700	700	173
		63	10	750	750	158
		67	10	800	800	144
		68	10	850	850	144
		77	10	900	900	129
		78	10	950	950	129
		78	10	1000	1000	129
		78	10	1050	1050	115
		78	10	1100	1100	115
		79	10	1150	1150	115
		82	11	1200	1200	100.625
	plants200					
		2089	1483	450	498.625	7504
		2078	1445	500	500	7503.75
		2103	1424	550	550	6483
		2189	1479	600	600	5778.75
		2217	1487	650	650	5189
		2225	1488	700	700	4945
		2220	1478	750	750	4585.63
		2209	1422	800	800	4370
		2276	1469	850	850	4183
		2243	1428	900	900	3967.5
		2221	1391	950	950	3838.13
		2254	1402	1000	1000	3780.63
		2285	1423	1050	1050	3665.63
		2306	1436	1100	1100	3565
		2313	1437	1150	1150	3479
		2443	1464	1200	1200	3464
	2n2-linear					
		23	7	450	450	86.25
		24	7	500	500	86.25
		24	7	550	550	71.875

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		24	7	600	600	71.875
		28	7	650	650	71.875
		28	7	700	700	57.5
		27	7	750	750	57.5
		28	7	800	800	57.5
		32	7	850	850	43
		32	7	900	900	43
		32	6	950	950	43
		32	6	1000	1.000	43.125
		32	7	1050	1050	43.125
		32	7	1100	1100	43.125
		32	7	1150	1150	43.125
		33	7	1200	1200	43
	simple-brick					
		38	10	450	450	100.625
		38	9	500	500	100.625
		40	9	550	550	86
		41	9	600	600	86
		43	9	650	650	71.875
		42	9	700	700	71.875
		49	9	750	750	58
		48	9	800	800	58
		48	9	850	850	58
		48	9	900	900	58
		48	9	950	950	57.5
		49	9	1000	1.000	58
		49	10	1050	1.050	57.5
		48	9	1100	1.100	57.5
		48	9	1150	1150	58
		58	9	1200	1.200	43.125
	columns					
		509	62	450	450	5326.63
		525	62	500	500	5240.38
		516	61	550	550	5168.5
		525	62	600	600	5104
		524	63	650	650	5054
		537	62	700	700	5010.25
		537	61	750	750	4974
		534	62	800	800	4946
		539	61	850	850	4917

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		544	61	900	900	4895
		560	62	950	950	4874
		577	61	1000	1000	4845
		600	62	1050	1.050	4831
		610	62	1100	1.100	4816
		611	62	1150	1.150	4802
		645	63	1200	1.200	4788
	counterfeit					
		204	53	450	779.25	1221.88
		203	52	500	779.25	1221.88
		202	52	550	779.25	1222
		203	52	600	779	1222
		202	52	650	779.25	1222
		203	52	700	779.25	1222
		202	52	750	779	1222
		199	49	800	800	1093
		207	50	850	850	906
		207	50	900	900	819
		218	53	950	950	762
		218	51	1000	1000	733.125
		250	52	1050	1050	690
		229	51	1100	1100	647
		228	51	1150	1150	633
		229	53	1200	1200	589
	diagonal5					
		25	11	450	450	273.125
		26	10	500	500	244.375
		27	10	550	550	230
		28	11	600	600	215.625
		28	10	650	650	186.875
		28	10	700	700	172.5
		29	10	750	750	172.5
		30	10	800	800	158.125
		30	10	850	850	143.75
		30	11	900	900	143.75
		32	10	950	950	143.75
		32	10	1000	1000	129.375
		34	10	1050	1050	129.375
		33	10	1100	1100	129.375
		35	10	1150	1150	115

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		35	11	1200	1200	115
ICBM (2)						
	cs-schedule					
		53	14	449.625	449.625	273
		76	15	491.125	491.125	244
		58	14	491.125	491.125	244.375
		57	14	558.375	558.375	201.25
		65	15	626.75	626.75	201
		68	15	664.375	664.375	187
		66	14	709.75	709.75	186.875
		68	14	785.25	785.25	143.75
		68	14	785.25	785.25	143.75
		67	14	785.25	785.25	143.75
		68	14	939.375	939.375	129.375
		67	14	939.375	939.375	129.375
		68	14	939.375	939.375	129.375
		68	14	939.375	939.375	129.375
		68	14	939.375	939.375	129.375
		90	15	1177.75	1177.75	115
	multipara					
		55	10	429.25	429.25	273
		57	9	472.125	472.125	244.375
		57	10	549	549	216
		57	10	549	549	215.625
		64	10	630	630	186.875
		65	10	699.25	699.25	172.5
		64	10	699.25	699.25	173
		69	10	775	775	158
		67	10	775	775	158
		78	10	851	851	144
		78	9	925.75	925.75	129.375
		79	9	925.75	925.75	129.375
		78	10	925.75	925.75	129.375
		78	9	925.75	925.75	129.375
		79	9	1102.75	1102.75	115
		80	10	1102.75	1102.75	115
	plants200					
		2071	1461	498.625	498.625	7503.75
		2093	1457	499	499	7503.75
		2104	1423	549.25	549.25	6483.13

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		2171	1458	599	599	5778.75
		2205	1475	650	650	5189.38
		2232	1489	699.125	699.125	4945
		2227	1484	748.625	748.625	4600
		2245	1453	799.875	799.875	4370
		2268	1459	846.75	846.75	4197.5
		2255	1437	895	895	3996.25
		2218	1388	949.25	949.25	3852.5
		2263	1412	992	992	3780.63
		2268	1409	1044.25	1044.25	3665.63
		2278	1410	1092.88	1092.88	3579.38
		2295	1425	1143.63	1143.63	3493.13
		2299	1424	1194.88	1194.88	3478.75
	2n2-linear					
		24	7	442.625	442.625	100.625
		24	6	494.875	494.875	86.25
		23	6	494.875	494.875	86.25
		24	6	494.875	494.875	86.25
		27	6	613.25	613.25	71.875
		27	6	613.25	613.25	71.875
		28	7	613.25	613.25	71.875
		28	7	613	613	71.875
		32	6	819	819	58
		32	6	819	819	57.5
		32	6	819	819	58
		32	6	819.25	819.25	58
		32	6	819.25	819.25	58
		33	7	819.25	819.25	57.5
		32	6	819.25	819.25	57.5
		32	6	819.25	819.25	58
	simple-brick					
		37	9	418.25	418.25	115
		37	9	418.25	418.25	115
		40	9	511.25	511.25	101
		40	9	511.25	511.25	100.625
		42	9	602	602	86.25
		42	9	602	602	86.25
		48	9	739.5	739.5	71.875
		48	9	739.5	739.5	71.875
		48	9	739.5	739.5	71.875

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		49	9	739.5	739.5	71.875
		47	9	739.5	739.5	71.875
		50	10	739.5	739.5	71.875
		48	9	739.5	739.5	71.875
		49	9	739.5	739.5	71.875
		48	9	739.5	739.5	71.875
		58	9	1159.13	1159.13	43.125
	columns					
		509	61	449.25	449.25	5326.63
		531	64	494.875	494.875	5247.5
		517	62	549	549	5168.5
		523	62	599.125	599.125	5103.75
		523	61	649.25	649.25	5054
		535	62	697.75	697.75	5010.25
		538	62	747	747	4974.38
		538	62	796.5	796.5	4946
		538	61	838.5	838.5	4924
		544	61	885.625	885.625	4902.5
		559	61	940.75	940.75	4873.75
		576	61	989.125	989.125	4852
		599	61	1048.25	1048.25	4831
		606	62	1095.25	1095.25	4816
		612	62	1128.88	1128.88	4809.13
		641	62	1171.88	1171.88	4794.75
	counterfeit					
		200	50	779.25	779.25	1222
		200	49	779.25	779.25	1221.88
		199	50	779.25	779.25	1221.88
		199	50	779.25	779.25	1221.88
		201	50	779.25	779.25	1222
		199	49	779.25	779.25	1222
		201	50	779.25	779.25	1221.88
		201	51	795.625	795.625	1121.25
		204	49	848.125	848.125	920
		207	50	897.625	897.625	833.75
		214	50	934.625	934.625	791
		218	50	987	987	733
		223	50	1043.25	1043.25	690
		251	50	1086	1086	661
		240	51	1139.75	1139.75	633

Algorithm	Sample	t	t_{ex}	$w_{\max}$	w	h
		226	50	1169.88	1169.88	618
	diagonal5					
		25	11	427.375	427.375	288
		25	10	493	493	244.375
		27	10	529.625	529.625	230
		27	10	564.125	564.125	230
		29	11	642.375	642.375	201.25
		28	10	642.375	642.375	201.25
		28	10	714	714	172.5
		29	10	794.125	794.125	158.125
		30	10	794.125	794.125	158.125
		29	10	794.125	794.125	158.125
		32	10	911.125	911.125	143.75
		32	10	911.125	911.125	143.75
		34	11	1041.13	1041.13	129.375
		35	11	1041.13	1041.13	129.375
		34	10	1041.13	1041.13	129.375
		34	11	1041.13	1041.13	129.375

Acknowledgements

Many people have contributed to this thesis and I want to take the time and space to thank them all for what they have given me.

Without Hans-Dieter Burkhard and Irfan Essa, this thesis would not exist. I thank them both for being my advisors under what have certainly been unusual circumstances. I am especially thankful for the trust they have had in this work and in me, and for the helpful and insightful comments I have received over the years both on my talks and on the many drafts of this thesis.

I am indebted in many ways to Markus and Arno. They sponsored this research, gave me ample time and resources to finish this work, and they supplied the necessary amounts of excellent coffee. They have made me an offer I could not refuse, twice, and they have given me endless opportunities to learn, through this thesis and my work. Most importantly, they have been friends. Markus had to use and endure every single prototype of the ICBM system over the years. This subtle pressure, but also Markus' feedback and his suggestions helped tremendously. Without Arno this work would not have been possible. He taught me programming and to this day he tries to teach me how to think, a challenge that frequently exceeds even his capabilities.

Arno and Markus founded think-cell Software in 2001. They have made this a great place to work, mostly by hiring great people. I want to thank Volker, for being a great office-mate, for always having an open ear, for the endless discussions on everything technology, for his ideas on user interface design, and for every piece of chocolate. Björn has done more for this thesis than he may know. He tested every last part of the ICBM system and more than once a seemingly small bug that he discovered made me rethink some big concepts. Valentin has become an invaluable colleague whose sound thinking has complemented me perfectly. Claudia contributed many great coffee breaks and her excellent graphic design work. Simon, Edgar, Vadim, Peter, Martin, Alex and Wasinee have kept everything that wasn't related to the ICBM system off my back. I also want to thank Mario Vukelic for his detailed and thoughtful user feedback.

I have been offered such opportunities because I have had the chance to learn from great teachers. Professor Starke's first-year course on logic, computability, and automata introduced me to the most fundamental aspects of computer science. Anusch Taraz' lessons on algorithms, graphs, and combinatorial optimization formed my way of thinking. Louchka Popova-Zeugmann taught me the basics of linear programming which turned out to be very useful. Once more, I would like to thank Hans-Dieter Burkhard for giving me the chance to work as a research assistant in his artificial intelligence workgroup, where I enjoyed Gabriela Lindemann-v.Trzebiatowski's (menthol!) cigarettes together with lively discussions. Several people whom I've never met personally were still kind enough to support my work in different ways. Kim Marriott, Peter Moulder, and Nathan Hurst provided me with the implementations of their table layout algorithms, and sample tables, both of which I have used in the evaluation chapter of this thesis. John Forrest, formerly of IBM, has developed the CLP solver that is used by the ICBM system. He has been incredibly responsive whenever I had a question about CLP's usage, he fixed bugs over the weekends, and even made changes to the solver that probably helped no one but me.

I would not be who I am today without my family who has shown me that hard work is a necessary condition of success and that it always pays off. I am most grateful to my parents, who have always believed in me and who have loved me unconditionally, who pushed me when I needed pushing and who comforted me when I needed comforting. I am very thankful to my father for teaching me math and generally doing more of the pushing, and to my mother, for her endless trust and for being my moral rolemodel.

Finally, I want to thank my friends who have made my life so much richer. Katrin and Wolfram have been wonderful hosts for the two weeks in which I started to write down this thesis. With Theresa and Thomas I have spent a fun week at the sea in our little PhD work camp. I thank Jan, Dirk, Sebastian, and Mareike for sharing their experiences as PhD students with me and for being my motivation when everyone but me had finished their thesis. Sascha, Tina, Henning, and Jessica deserve my gratitude for working much more than I did, so I could never complain, for keeping my mood up on the weekends, and for every barbecue. Dana, Stefan, Jule, Friedemann, and Sophie, my oldest friends, still invited me, although I didn't call for months. I thank Edna for the spirited discussions and disputes we have had together in matters big and small, and I want to thank Micky for being the best friend I have, for teaching me the basics of economics, and for pretending, year after year, to be interested in my work while he listened to my boring explanations of what I actually do. I thank Ines for her infinite patience and support in stressful times. You rock!

Selbständigkeitserklärung

Ich erkläre hiermit, dass

- ich die vorliegende Dissertationsschrift “Sketching Slides – Interactive Creation and Automatic Solution of Constrained Document Layout Problems” selbständig und ohne unerlaubte Hilfe angefertigt habe,
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze,
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist, gemäß amtlichem Mitteilungsblatt Nr. 34/2006

Ort, Datum	Unterzeichner